



**Konzept und Ansatz einer Wertschöpfungskette für die Erkennung und  
Bereitstellung neuer Fahrumfänge für intelligente Fahrzeuge**

Bachelorarbeit

An der  
Carl von Ossietzky Universität Oldenburg  
Studiengang Wirtschaftsinformatik

Vorgelegt von Linus Hestermeyer  
*linus.hestermeyer@gmail.com*  
Matr.Nr.: 4087097

Erstprüfer: Prof. Dr. Frank Köster  
Zweitprüfer: Dipl.-Inform. Gerald Sauter

Oldenburg, den 21. Juni 2020



# Vorwort

Diese Arbeit entstand im Rahmen meines Wirtschaftsinformatik Studiums an der Carl von Ossietzky Universität Oldenburg.

Mein Besonderer Dank gilt meinen Betreuern Gerald Sauter und Prof. Dr. Frank Köster für die stetige fachliche Betreuung und Hilfe bei der Erstellung der Arbeit und des Prototypen. Der initiale Gedankenaustausch über die Thematik und die gemeinsame Anpassung des Kernthemas der Arbeit haben das Ergebnis erst möglich gemacht.

Selbstverständlich gilt mein Dank auch meinen Eltern Andrea und Andreas sowie meinen Geschwistern Johanna und Lukas, die mich bei der Entwicklung meiner fachlichen und beruflichen Laufbahn stets unterstützt haben.

Nicht zuletzt möchte ich meiner Freundin Julia, der Fachschaft Informatik der Universität Oldenburg und dem OFFIS danken, welche während des Studiums immer eine gute Anlaufstelle für fachliche aber auch persönliche Aspekte meiner selbst gewesen sind.

Oldenburg, im Juni 2020

Linus Hestermeyer

# Inhaltsverzeichnis

<b>1 Motivation</b>	<b>1</b>
<b>2 Theoretischer Rahmen</b>	<b>2</b>
2.1 Einführung . . . . .	2
2.2 Bedarfserkennung von Software . . . . .	3
2.2.1 Relevante Systeme eines autonomen Fahrzeugs in der Bedarfserkennung . . . . .	3
2.2.2 Eine geeignete Kommunikationsgrundlage: OpenSCENARIO . . . . .	4
2.2.3 Software-User-Pattern-Recognizer [SUPR] im Rahmen der Bedarfserkennung	6
2.3 Bereitstellung von Software . . . . .	7
2.3.1 OTA-Aktualisierungen mit UPTANE . . . . .	8
2.3.2 Anpassung von UPTANE Architektur . . . . .	9
2.3.3 SUPR im Rahmen der Softwarebereitstellung . . . . .	11
2.3.4 Die Mensch-Maschine-Schnittstelle . . . . .	15
2.4 Technologie- und Methodik-Scouting . . . . .	16
2.4.1 Arbeitsmethodik: User-Centered-Design . . . . .	16
2.4.2 Tech-Scouting . . . . .	19
2.5 Ausblick und erstes Konzept der praktischen Umsetzung . . . . .	20
2.6 Eingrenzung des Themas . . . . .	23
<b>3 Das Business Model Canvas und die Wertschöpfungskette</b>	<b>24</b>
3.1 Business Model Canvas . . . . .	24
3.1.1 Kundensegmente . . . . .	25
3.1.2 Kundenbeziehungen . . . . .	25
3.1.3 Marketingkanäle . . . . .	25
3.1.4 Nutzenversprechen . . . . .	26
3.1.5 Einnahmequellen . . . . .	27
3.1.6 Schlüsselressourcen . . . . .	27
3.1.7 Schlüsselaktivitäten . . . . .	27
3.1.8 Schlüsselpartner . . . . .	28
3.1.9 Kostenstruktur . . . . .	29
3.2 Die Wertschöpfungskette . . . . .	30
3.2.1 Primäre Aktivitäten . . . . .	30
3.2.2 Unterstützende Aktivitäten . . . . .	35
3.3 Zusammenfassung . . . . .	37
<b>4 Technische Konzepte</b>	<b>38</b>
4.1 Klassifizierung von Software . . . . .	38
4.1.1 Berechtigungen . . . . .	38
4.1.2 Softwarekennzahlen . . . . .	39
4.2 Umgebungsbedingte Softwaresuche . . . . .	40
4.3 Kommunikationsprotokolle . . . . .	40
4.3.1 Selbstständiger Softwarekauf . . . . .	41
4.3.2 Installationsvorschlag vom Service Provider . . . . .	42
4.3.3 Nutzung eines Service . . . . .	42
4.4 Ausblick . . . . .	45

<b>5 Vorstellung des Prototypen</b>	<b>46</b>
5.1 Architektur des Prototypen . . . . .	46
5.1.1 Architektur des Software Applikation Managers (SAM) . . . . .	47
5.1.2 Architektur des OEM-Servers (Server) . . . . .	48
5.2 Installation . . . . .	49
5.2.1 Server und SAM Installation . . . . .	49
5.2.2 MMS Installation . . . . .	49
5.2.3 Carla Installation . . . . .	50
5.3 Nutzung des Prototypen . . . . .	51
5.4 Analyse des Prototypen und Ausblick auf die Weiterentwicklung . . . . .	53
<b>6 Reflexion der Ergebnisse</b>	<b>55</b>
<b>7 Ausblick</b>	<b>55</b>
<b>8 Codeentnahmen</b>	<b>59</b>

## Abbildungsverzeichnis

1	Architektur AV nach Jeff Schneider (CMU) [5] . . . . .	4
2	Ausschnitt einer OpenSENARIO Datei von Andreas Biehn [8] . . . . .	5
3	Umweltanalyse . . . . .	7
4	Architektur Design UPTANE [7] . . . . .	8
5	Angepasste Uptane Architektur . . . . .	10
6	Technisches Erstkonzept . . . . .	22
7	Supply Chain von Software . . . . .	30
8	Die wichtigsten Bausteine einzelner Aktivitäten der Wertschöpfungskette . . . . .	31
9	Jeff Schneider: Architecture of Autonomous Vehicles [5] . . . . .	38
10	Eigenständige Installation von Software . . . . .	41
11	Installationsprozess von Software . . . . .	43
12	Nutzungsprozess von Software . . . . .	44
13	Grundlegende Architektur der Prototypen . . . . .	46
14	Aufbau der Common Library . . . . .	47
15	Architektur des Software Applikation Managers . . . . .	48
16	Architektur des Servers . . . . .	48
17	Steuerung des Prototypen über die GUI . . . . .	52

# 1 Motivation

Das autonome Fahren ist die Zukunft des Automobils. Wann Fahrzeuge jedoch tatsächlich vollständig fahrerlos Fahren ist noch nicht absehbar. Künftig entwickelte Softwares können Fahrzeugen einzelne autonome Fahrfunktionen hinzufügen, durch welche diese immer selbstständiger fahren können. Damit Softwares auch nach dem Kauf eines Fahrzeugs vom Fahrzeughalter installiert werden können, soll der Kauf und die Installation dieser über eine kabellose Schnittstelle möglich sein. Dies kann durch die Integration eines Software-Shops in das Fahrzeug realisiert werden über welchen Fahrzeughalter eigenständig auswählen können welche Softwares auf diesem installiert werden sollen. Um dies verwirklichen zu können muss festgestellt werden, *was relevante Bausteine einer Wertschöpfungskette für die Erkennung und Bereitstellung neuer Fahrumfänge für intelligente Fahrzeuge sein können und wie Automobilhersteller diese umsetzen könnten*. Die Ergebnisse der Forschungsfrage sollen in einem Prototypen veranschaulicht werden, welcher den Kauf, die Installation und Nutzung einer Software darstellt.

Zunächst werden in Kapitel 2 Grundlagen, bestehende Technologien und eigens erstellte System-Konzepte vorgestellt, welche der Bedarfserkennung und Bereitstellung von Software zuzuordnen sind und diese unterstützen. In Kapitel 3 wird anhand eines Business Model Canvas ein Überblick des Marktes für Fahrfunktionssoftware geschaffen und die in diesem identifizierten Schlüsselaktivitäten werden daraufhin in einer Wertschöpfungskette geordnet und erläutert. In Kapitel 4 wird ein Konzept zur Klassifizierung von Software vorgestellt, nach der Art der unterschiedlichen Zugriffsrechte auf die Systeme des Fahrzeugs. Für die individuelle Bedarfsbestimmung von Software wird ein Suchalgorithmus skizziert und es werden Kommunikationsprotokolle erstellt, welche die Interaktion zwischen dem Server, einem Fahrzeug und anderen Akteuren der Umwelt darstellt. In Kapitel 5 wird schließlich der Prototyp vorgestellt, welcher die Ergebnisse der vorigen Kapitel visuell veranschaulichen soll.

*Das folgende Kapitel wurde während des vorangegangenen Forschungsseminars erstellt.*

## 2 Theoretischer Rahmen

### 2.1 Einführung

"Durch den Einsatz von Elektrik und Elektronik ist das Automobil in den vergangenen Jahrzehnten stark geprägt worden, und die „Intelligenz“ in den Subsystemen hat exponentiell zugenommen." (Vgl. [4, S. 1]) Mit zunehmend mehr intelligenten Fahrassistenten in modernen Fahrzeugen ist es Absehbar, dass bereits in naher Zukunft Fahrzeuge die Stufe 3 bzw. teilweise Stufe 4 des Autonomen Fahrens erreichen. Ein Fahrzeug der Stufe 3 ist dazu in der Lage dazu, die Längs- und Querlenkung in bestimmten Anwendungsfällen selber übernehmen und diese so sicher zu durchfahren. Hierbei wäre allerdings noch ein Insasse notwendig, der in einem Notfall das Steuer übernehmen kann. In Stufe 4 Fahrzeugen kann das Fahrzeug die komplette Fahraufgabe in bestimmten Anwendungsfällen übernehmen. [25, S. 14]

Erst Stufe 5 Fahrzeuge werden "vollumfänglich auf allen Straßentypen, in allen Geschwindigkeitsbereichen und unter allen Umfeldbedingungen die Fahraufgabe vollständig allein durchführen. Wann dieser Automatisierungsgrad erreicht sein wird, kann heute noch nicht benannt werden." (Vgl. [25, S. 14]). Aufgrund dessen ist es wichtig, dass Stufe 3 und Stufe 4 Fahrzeuge in Zukunft mehr Anwendungsfälle abdecken können. Hierzu sollen diese über eine kabellose Schnittstelle ortsunabhängig Softwarepakete herunterladen können, welche das Spektrum der autonomem Fahrfunktionen des Fahrzeugs zweckgebunden erweitert. Angenommen Sie planen mit ihrem neuen Fahrzeug eine Autofahrt nach England. Spätestens ab dem Ende des Eurotunnels wäre es für das Fahrzeug nicht mehr möglich selbstständig zu fahren, da dort Linksverkehr herrscht. Das Auto soll automatisch erkennen können, dass es ab einem bestimmten Punkt nicht mehr selbstständig fahren kann. In Folge dessen soll es eine Software zum Kauf/Miete anbieten, welche es dem Auto ermöglicht, am Linksverkehr teilnehmen zu können. Die Halter können ihr Fahrzeug dementsprechend zunehmend autonom fahren lassen, was den Marktwert des Autos nach dem Kauf steigern kann. Die Fahrzeughalter sollen bei Kauf von Software vom System unterstützt werden in Form von Vorschlägen für neue Software, die den Bedarf des Fahrers abdeckt. Diese Vorschläge sollen zu Zeitpunkten erfolgen, in denen der Verkauf einer bestimmten Software möglichst wahrscheinlich ist. Mittels dessen wird zudem unterbunden, dass der Nutzer von zu vielen Benachrichtigungen überfordert wird.

Damit Anwendungsfälle rasch abgedeckt werden können, bedarf es einem großen Spektrum an Software, die eben diese abdeckt. Um dies schnellstmöglich bewerkstelligen zu können ist es erforderlich, dass die Entwicklung von Softwarepakete durch Zulieferer geschehen, welche sich explizit auf diesen Markt fokussieren. Der hierdurch entstehende Seitenmarkt für die Entwicklung von Fahrfunktionssoftware kann somit schnell wachsen und sich als Teil in der Automobilindustrie etablieren. Durch autonome Fahrfunktionen werden Autos in der Regel schonender gefahren als vom Menschen, weshalb die Lebenszeit von Autos voraussichtlich verlängert wird. [25, (S. 16)] Ericssons Juergen Daunis sagt diesbezüglich, dass die meisten Analytiker und Führungskräfte der Meinung sind, dass der Umsatz dieser neu entstehenden Seitenmärkte weiter steigen wird und dass das traditionelle Geschäftsmodell an dem wirtschaftlichen Maximum seiner Existenz ist. [6]

Die in dieser Arbeit erfassten Gliederungspunkte dienen als Grundlage für die gleichnamige Bachelorarbeit und sind im Wesentlichen als Literaturrecherche und Grundlagenerarbeitung zu verstehen. Es werden einzelne Bausteine der in der Bachelorarbeit zu erstellenden Wertschöpfungskette erstmalig benannt und konkretisiert. In Kapitel zwei wird dargestellt, wie das Auto einen Softwarebedarf erkennt, wie es einen Server hierüber informiert und wie dieser Server die richtige Software sucht. In Kapitel Drei wird zum einen eine sichere Architektur für kabellose Aktualisierungen erarbeitet. Des weiteren wird erläutert, wie Software verkauft wird und es werden Richtlinien für die Mensch-Maschine-Schnittstelle erarbeitet, auf welcher ein Angebot letzten Endes angezeigt werden soll.

## 2.2 Bedarfserkennung von Software

Im Rahmen der Bedarfserkennung steht die Beschreibung der eigenen Umgebung von Fahrzeugen im Mittelpunkt. Damit die Softwarehersteller den Bedarf von Fahrzeugen abdecken können, muss ihnen die Umgebung des Fahrzeugs zu dem Zeitpunkt der Bedarfserkennung bekannt sein. Um zu verstehen, wie ein Fahrzeug seine Umwelt eigens beschreiben kann, wird im Folgenden die Architektur eines autonomen Fahrzeugs vorgestellt und erläutert, welche Bestandteile dessen für die Bedarfserkennung relevant sind.

### 2.2.1 Relevante Systeme eines autonomen Fahrzeugs in der Bedarfserkennung

Um die Suche nach Software zu erleichtern, benötigt die Suche auf dem Server gewisse Daten als Input. Damit die Situation, in welcher das Fahrzeug nicht selbstständig fahren kann, durch eine Software abgedeckt werden kann ist es wichtig eine Beschreibung der Umwelt des Autos zu dem Zeitpunkt der Bedarfserkennung zu erstellen. Durch die Fusion aufgenommener Kamera-, Ultraschall- und Radardaten kann die Umwelt beziehungsweise die Umgebung des Autos in einem Datenformat wie dem von der ASAM.<sup>1</sup> definierten Standard "OpenSCENARIO" [8] dargestellt werden. Dieser wird in Kapitel 2.2.2 vorgestellt.

Im Rahmen der Suche nach möglicherweise nötiger Software spielt die Routen- und Bewegungsplanung des Autos eine große Rolle. Zum einen kann das Auto bei der Eingabe einer neuen Route diese nach Gegenden absuchen in denen oft neue Software benötigt wird. Es kann infolgedessen diese dem Fahrer bereits vor Fahrtbeginn vorschlagen und somit die Bequemlichkeit der Fahrt sicherstellen. Zum anderen soll das Auto Muster im Fahrtenverlauf erkennen, um so bei der Erkennung eines Softwarebedarfs auf einer dieser Strecken eben diese Software zum Kauf vorzuschlagen.

Abbildung 1 zeigt die Architektur eines Autonomen Fahrzeugs nach Jeff Schneider. Sie verdeutlicht, welche Bestandteile ein Autonomes Fahrzeug besitzt um mit Hilfe dieser selbstständig zu fahren. Die aufgenommenen *Sensordaten* leiten Informationen an die *Karten- & Positionsverfolgung* weiter. Durch einen Merge (Zusammenführung) dieser Daten kann das Fahrzeug die eigene Umwelt *wahrnehmen* und mit Hilfe der dynamischen Inhalte der Welt (z.B. Geschwindigkeiten) *Vorhersagen* für den Verkehr stellen. Als Bündel stellen sie die Bewegungsplanung dar.

---

<sup>1</sup><https://www.asam.net/standards/detail/openscenario/>

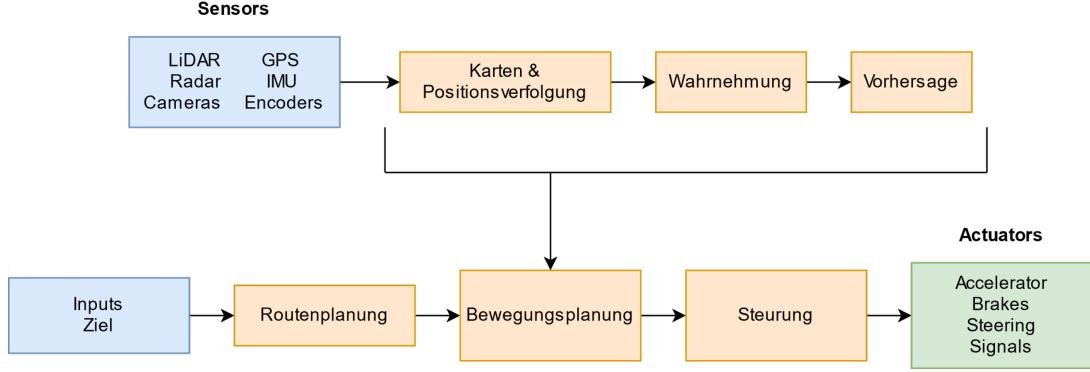


Abbildung 1: Architektur AV nach Jeff Schneider (CMU) [5]

Der Fahrer gibt dem Auto initial ein *Ziel*, mit Hilfe dessen das Fahrzeug die zu fahrende Route berechnet. Die Bewegungssteuerung gibt das Wahrgenommene und Vorhergesagte weiter an die Steuerung welche letztendlich entscheidet, was die einzelnen Aktuatoren machen. Damit der Bedarf einer Softwarelücke festgestellt werden kann, müssen die Systembausteine "Sensoren", "Karten & Positionsverfolgung", "Wahrnehmung" und "Vorhersage" angeknüpft werden um so herauszufinden **wann** das Fahrzeug nicht mehr selbstständig fahren kann. Wurde dieser Moment festgestellt, wird die Übergabe der Fahraufgabe initiiert und zeitgleich auch die Suche nach Software gestartet. Neben den wichtigen Bausteinen der Bewegungsplanung kann auch anhand der Routenplanung von Fahrzeugen ein Softwarebedarf untersucht werden (Siehe 2.2.3). Zunächst wird eine Möglichkeit dargestellt, wie die Daten der Bewegungsplanung in einer geeigneten Form gespeichert und versendet werden können.

### 2.2.2 Eine geeignete Kommunikationsgrundlage: OpenSCENARIO

Der Inhalt dieses Kapitels basiert auf Inhalten der Projektwebseiten des OpenSCENARIO Standards. [8].

OpenSCENARIO ist ein XML-basiertes Dateiformat zur Beschreibung aller statischen (*Gegenstände, Hindernisse, etc.*) und dynamischen (*Geschwindigkeiten, Bewegungsrichtungen, etc.*) Inhalte der Umwelt eines Fahrzeugs. "Der primäre Anwendungsfall von OpenSCENARIO ist es komplexe, synchrone Manöver zu beschreiben, welche mehrere Entitäten wie Fahrzeuge, Fußgänger und andere Verkehrsteilnehmer betreffen." (Vgl. [8, asam.net])

Abbildung 2 zeigt einen Ausschnitt einer OpenSCENARIO-Datei. Für ein Szenario ist immer das **Straßenetzwerk** (*RoadNetwork*) sowie die beinhalteten **Entitäten** (*Entities*) festzulegen. Das eigentliche Szenario ist innerhalb eines **Storyboards** dargestellt und unterteilt sich in einzelne **Storys** (Siehe Abbildung 2). Alle zuvor definierten Entitäten erhalten eine initiale **Action**, welche das initiale Handeln der Entität festlegt (Siehe <Init>-Block).

Eine einzelne Story ist immer einer einzelnen Entität zuzuordnen. Eine Story wiederum ist in **Äcts** aufgeteilt, welcher eine Sammlung an **SSequenzes** beinhalten kann. Jedes dieser Elemente kann mit einer **"Condition"** versehen werden. Wird die "Condition" (Bedingung) erfüllt, wird der jeweilige Story-/Act- oder Sequenz-Block ausgeführt.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <OpenSCENARIO>
3
4  <FileHeader revMajor="0" revMinor="9" date="2017-02-24T10:00:00"
5    description="Sample Scenario - Overtaker" author="Andreas Biehn"/>
6
7  <ParameterDeclaration/>
8
9  <Catalogs>
38
39  <RoadNetwork>
40    <Logics filepath="Databases/PEGASUS/PEGASUS_A01.xodr"/>
41    <SceneGraph filepath="Databases/PEGASUS/PEGASUS_A01.opt.osgb"/>
42  </RoadNetwork>
43
44  <Entities>
45    <Object name="Ego">
51    <Object name="A1">
57  </Entities>
58
59  <Storyboard>
60    <Init>
61      <Actions>
62        <Private object="Ego">
79        <Private object="A1">
96      </Actions>
97    </Init>
98    <Story name="MyStory" owner="A1">
99      <Act name="MyAct">
100     <Sequence name="MySequence" numberOfExecutions="1">
175     <Conditions>
186     </Act>
187   </Story>
188   <End>
189   <End>
190 </Storyboard>
191
192 </OpenSCENARIO>

```

Abbildung 2: Ausschnitt einer OpenSENARIO Datei von Andreas Biehn [8]

OpenScenario ist aus mehreren Gründen gut geeignet um als Kommunikationsgrundlage zwischen Auto und Server zu fungieren. Zum einen handelt es sich dabei um eine Opensourcelösung, wodurch jeder Entwickler Weltweit selbstständig mit diesem Arbeiten kann. Zum anderen existiert wegen dem XML-basierten Dateiformat eine gute Lesbarkeit und Regelmäßigkeit Abfolge der Szenarien. Ein Szenario kann aufgrund dieser Regelmäßigkeit einfach und schnell erstellt werden. Der letzte und größte Vorteil von OpenScenario ist es, dass man ein Szenario in dem Opensource Simulator "Carla" abspielen kann.

Der im Folgenden vorgestellte Software-User-Pattern-Recognizer (SUPR) nutzt das OpenScenario-Format zur Suche nach Software. Welche Suchvarianten es gibt und weitere Aufgaben des SUPR werden hier vorgestellt.

### **2.2.3 Software-User-Pattern-Recognizer [SUPR] im Rahmen der Bedarfserkennung**

Das in diesem Abschnitt vorgestellte Konzept des Software-User-Pattern-Recognizer (kurz: SUPR) ist ein eigens ausgearbeiteter Baustein, welcher die Bedarfserkennung für intelligente Fahrzeuge beschleunigen und vereinfachen soll. Die Aufgaben des SUPR lassen sich in Aspekte der Bedarfserkennung sowie der Bereitstellung aufteilen, weshalb der SUPR in Kapitel 3.3 wieder aufgegriffen wird.

Eine Aufgabe des SUPR im Rahmen der Bedarfserkennung ist es, auf Anfrage hin eine Suche nach passenden Softwarepaketen durchzuführen (**1.**). Die zweite Aufgabe ist die regelmäßige Suche nach Software in der Umwelt/Region des Autos bzw. die Suche nach Softwarepaketen entlang zu Fahrenden Strecken (**2.**). Das Maß der Rechenleistung auf Seiten des Servers ist indes höher als auf dem Fahrzeug. Eine mögliche Gegenüberstellung von Rechenleistung und Sendeleistung des Servers und des Autos kann zu behebende Defizite der Architektur aufdecken - die Optimierung des Systems ist jedoch nicht Teil des Forschungsseminars.

Der Server, auf welchem die Suche stattfindet, benötigt zwei Datenbanken: die eine enthält sämtliche Softwarepakete für intelligente Fahrzeuge, die andere eine große Menge an OpenScenario Dateien, welche zusätzlich Fremdschlüsselverweise auf ein oder mehrere Softwarepakete bereitstellen. Nur wenn diese Bedingung erfüllt ist, kann eine Suche durchgeführt werden.

#### **1. Gezielte Suchanfragen**

Gerät ein intelligentes Fahrzeug im Laufe seiner Fahrt in eine ihm unbekannte Situation, kann es diese mit Hilfe von OpenSCENARIO beschreiben. Das vom Auto erstellte Szenario wird an den Server geschickt, welcher dieses mit den auf der Datenbank liegenden Szenarien abgleicht. Ist die Suche abgeschlossen, wird eine Fallunterscheidung zwischen den folgenden Optionen getätigt:

##### **A Es wird kein passendes Softwarepaket gefunden**

Entweder existiert zu dem übergebenen Szenario keine Software oder möglicherweise wurde die im Szenario dargestellte Situation noch nicht zu einer Software gemapped. Dies sollte von speziell hierfür angestellten Arbeitnehmern überprüft werden, um so Dopplungen in der Softwareentwicklung zu vermeiden.

##### **B Es werden (mehrere) passende Softwarepakete gefunden.**

In diesem Fall gilt zu entscheiden, welches Softwarepaket die besten Auswirkungen auf die Performance des Autos hat. Hierzu ist das Heranziehen diverser Kennzahlen ratsam um somit die Performance der unterschiedlichen Softwares untereinander zu vergleichen und eine fundierte Entscheidung treffen zu können. Eine beispielhafte Ausarbeitung dieser Entscheidungstreffung stellt Niklas Stelter in seiner Bachelorarbeit vor [31]. Am Ende einer Suche soll entweder eine einzelne oder keine Software vorgeschlagen werden.

Neben der vom Fahrzeug ausgehenden Suche nach einer bestimmten Software, kann dieses auch Vorschläge für Software von einem Server erhalten, welcher dauerhaft die Umwelt intelligenter Fahrzeuge analysiert.

**2. Ortsbezogene Erkennung eines Softwarebedarfs** Neben der Suche nach einer bestimmten Software soll der SUPR auch Suchen in der Heimatregion des Autos durchführen sowie auf zu Fahrenden Strecken. Diese Suchen basieren nicht auf OpenScenario-Dateien sondern auf Basis der Routenplanung[A] (Siehe Abbildung 1) und der Fahrtenhistorie des Fahrzeugs[B].

#### **A: Suche auf Basis der Routenplanung**

Wird vom Fahrer eine nicht oft oder noch gar nicht zurückgelegte Strecke vorgegeben, so soll der

SUPR entlang der zu fahrenden Strecke häufig auftretenden Softwarebedarf identifizieren und dem Fahrer vor oder kurz nach Antritt der Reise diese Softwarevorschlägen. Daraus ergibt sich die Anforderung, zu jedem sich in der Datenbank befindlichen OpenScenario auch die geographische Position der Bedarfsentdeckung zu speichern. In Abbildung 3 ist folgende Situation dargestellt: Unser Auto hat als Ziel im Navigationssystem "Braunschweig" und befindet sich aktuell auf der A2. Die roten Boxen stellen den möglichen Bereich dar, in welchem der Server die Umweltanalyse betreibt. Wird eine Software entdeckt, überprüft das System die Relevanz für das eigene Fahrzeug und schlägt sie je nach dem vor oder nicht.

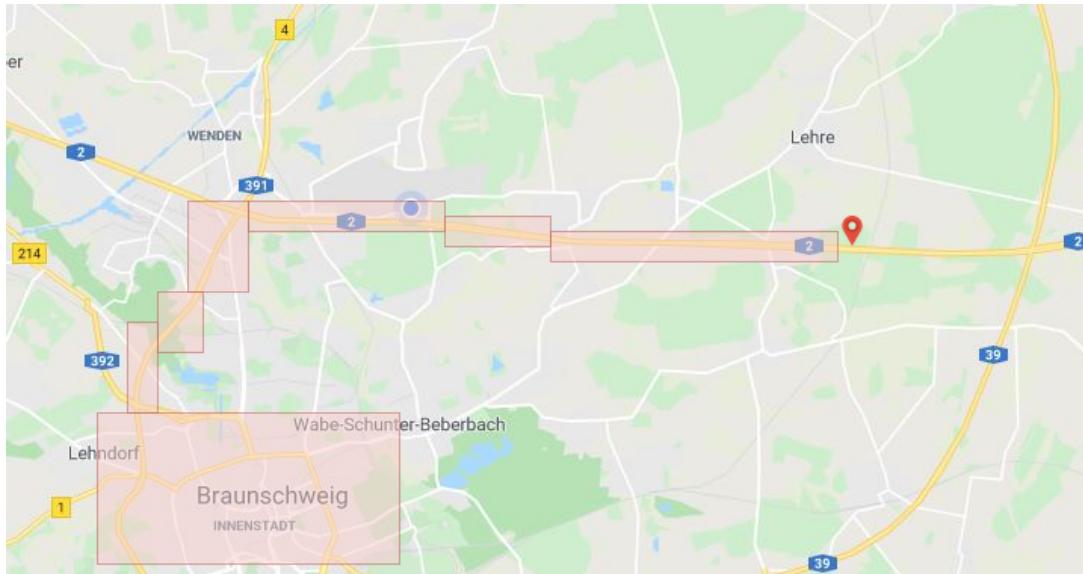


Abbildung 3: Umweltanalyse

#### **B: Suche auf Basis der Fahrtenhistorie**

Hierbei werden Muster in den zurückgelegten Strecken des Fahrers gesucht, um so dessen meist gefahrene Strecken zu identifizieren. Das Auto soll abschätzen können, wann der Fahrer welche Strecke zurücklegen wird. Vor dem jeweiligen Fahrtantritt soll die Suche nach Software identisch zu Variante A durchgeführt worden sein. Der Unterschied ist, dass die Suche hierbei selbstständig erfolgen soll, was ein besseres Nutzungserlebnis für den Fahrer schafft.

Ist die Suche (*egal ob gezielte Suche oder Umweltanalyse*) abgeschlossen, kann die Software dem Fahrer zum Kauf vorgeschlagen und bei Bestätigung für das Auto bereitgestellt werden. Das nächste Kapitel verdeutlicht den Umfang der Bereitstellung.

### **2.3 Bereitstellung von Software**

Wurde der Bedarf einer neuen Software vom Fahrzeug festgestellt, muss diese im folgenden bereitgestellt werden. Zuvor muss der Softwarezulieferer die Sicherheit von Softwarepaketen verifizieren und eine sichere Kommunikation zwischen Servern und Fahrzeugen herstellen. Der Server muss alle vorhandenen Softwarepakete verwalten und auf Anfrage das am besten geeignete Softwarepaket für das jeweilige Fahrzeug identifizieren und ein Angebot an dieses schicken. Das Fahrzeug muss die eingegangenen Softwareangebote über die Mensch-Maschine-Schnittstelle zu einem Zeitpunkt anbieten, an dem die Kaufbereitschaft des Fahrers möglichst hoch ist. Hierzu bedarf es einem Soft-

waremanagement, welches die Angebote des Servers verarbeitet. Da die Grundvoraussetzung der Wertschöpfungskette eine sichere Kommunikation zwischen Auto und Softwarelieferanten ist, wird zunächst eine mögliche Architektur zur sicheren Kommunikation vorgestellt.

### 2.3.1 OTA-Aktualisierungen mit UPTANE

Der im folgenden vorgestellte Standard "UPTANE", stellt eine Architektur und Vorgehensweise für die sichere Kommunikation zwischen einem Server und einem Auto vor. Hierdurch können Software-Aktualisierungen an intelligente Fahrzeuge sicher verteilt werden [7]. Erste Schritte hierzu wurden 2010 getätigt, als Justin Samuel, Nick Mathewson, Roger Dingledine und Justin Cappos "*The Update Framework*"(TUF) entwickelten. Dieses bildet den Grundbaustein für den späteren "UPTANE"-Standard. Mittlerweile ist es Teil des Automotive Grade Linux Projekts und somit Teil der "Linux Foundation". Uptane stellt mittels einer Mehrschichtenarchitektur sicher, dass im Rahmen von Aktualisierungen keine Schädlingssoftware auf ein Auto gelangt. Zunächst wird die Architektur von Uptane dargestellt (Abb. 4).

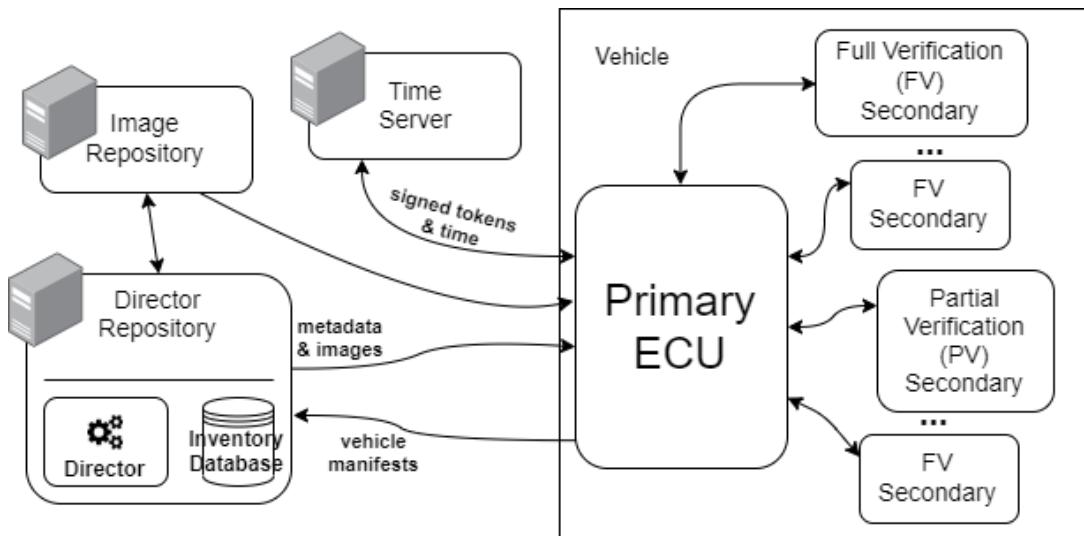


Abbildung 4: Architektur Design UPTANE [7]

Es ist vorab wichtig zu erwähnen, dass Uptane lediglich der Standard ist und keine offizielle Implementierung bereitstellt. Beispielhafte Implementierungen wären aktualizr<sup>1</sup>, rust-tuf<sup>2</sup>, Notary<sup>3</sup> oder die OTA Community Edition<sup>4</sup>. Kommerziell entwickelte Systeme werden bereitgestellt von HERE Technologies<sup>5</sup> sowie von Airbiquity<sup>6</sup>.

Die rechte Seite des Bildes stellt das Fahrzeug dar, die Elemente zur linken die Repositories. Diese Repositories sind Server und sie haben alle eine eigene wichtige Aufgabe. Der Time-Server ist dafür da, um ECUs über die aktuelle Zeit zu informieren, da viele ECUs keine Uhr haben [7]. Das Image Repository speichert jedes derzeit vom Lieferanten verteilte Image zusammen mit den Meta-Daten, welche zur Authentizität benötigt werden. Es nutzt Offline-Schlüssel um alle Metadaten zu unter-

<sup>1</sup><https://github.com/advancedtelematic/aktualizr>

<sup>2</sup><https://github.com/heartsucker/rust-tuf>

<sup>3</sup><https://github.com/theupdateframework/notary>

<sup>4</sup><https://github.com/advancedtelematic/ota-community-edition/>

<sup>5</sup><https://www.here.com/products/automotive/ota-technology>

<sup>6</sup><https://www.airbiquity.com/product-offerings/software-and-data-management>

schreiben" bzw. zu verifizieren, was einen hohen Sicherheitsvorteil darstellt. Das Director Repository entscheidet abhängig von den übergebenen Informationen des Autos genau, welche Images an die ECUs verteilt werden müssen. Ein Auto hat mehrere ECUs, welche sich in Speicherplatzgröße, Stromverbrauch und Aufgabenbereich unterscheiden. Die *Primary ECU* verwaltet und steuert die Installationen auf den anderen ECUs.

In dem ersten Schritt einer Aktualisierung schickt das Fahrzeug sein Manifest an das Director Repository. Dieses enthält Informationen darüber, welche Images aktuell auf dem Auto installiert sind. Das Director Repository entscheidet anhand dessen, welche Software auf dem Fahrzeug installiert/aktualisiert werden soll. Die Metadaten und die neuen Images werden zurück an die ECU geschickt. Hier findet zunächst eine Verifikation der neuen Images statt, bevor sie anschließend bei erfolgreichem Test installiert werden. Die Verifikation der ECUs kann entweder vollständig oder teilweise erfolgen. **Vollständig** heißt in diesem Kontext, dass die Größe der Software und die Hashees, welche die ECU über die Metadaten vom Director Repository erhält, identisch sind zu denen der Metadaten die vom Image Repository zur Verfügung gestellt werden. Bei der **teilweisen** Verifikation muss lediglich die Signatur der Metadaten des Director Repositories mit der Signatur der Metadaten vom Image Repository übereinstimmen.

### 2.3.2 Anpassung von UPTANE Architektur

Damit die durch UPTANE bereitgestellte Architektur die Erkennung und Bereitstellung neuer Fahrumfänge unterstützt, muss diese dementsprechend angepasst werden, damit neben dem Aktualisieren bereits installierter Software auch das Installieren neuer Software möglich ist. Hierzu wird die Architektur von UPTANE erweitert und angepasst.

Damit das Director Repository nicht nur bestimmen kann, welche Software eine Aktualisierung benötigt, sondern auch welche auf dem Fahrzeug installiert werden muss um den Bedarf zu decken, braucht es eine geeignete Kommunikationsgrundlage. Hierzu soll das zuvor vorgestellte OpenSCENARIO XML-Speicherformat verwendet werden. Um die Suche nach neuer Software für das Directory Repository zu ermöglichen, wird das in Abbildung 5 dargestellte *SScenario Repository*" eingeführt. In diesem werden OpenSCENARIO-Dateien gespeichert, welche als Basis für die Suche nach Software dienen. Jede Datei hat hält mindestens **eine** Referenz in Form eines Schlüssels auf eine Software aus dem Image Repository.

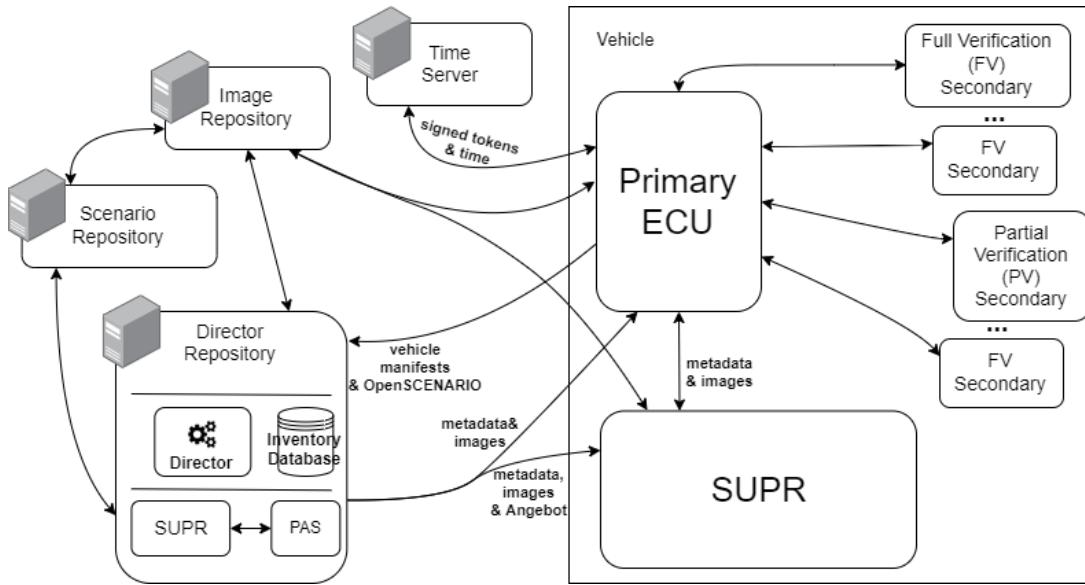


Abbildung 5: Angepasste Uptane Architektur

Dies bedeutet, dass die jeweilige Situation der OpenScenario-Datei von eben dieser Software bewältigbar ist. Erhält das Director Repository nun eine Anfrage für eine neue Software, wird diese anhand der mitgeschickten OpenSCENARIO-Dateien gesucht. Bei erfolgreicher Suche, soll diese in das Scenario Repository eingetragen werden. Die Suche führt der in Kapitel 2.3 eingeführte Software-User Pattern-Recognizer durch. In Kapitel 3.4 wird dieser um die Aspekte der Bereitstellung von Software erweitert.

Fahrzeughalter müssen neue Software kaufen, weshalb im Fall einer gefundenen Software zusätzlich zu den Metadaten und dem Softwarepaket (*Softwareimage*) noch eine Sammlung verkaufsbezogener Daten verschickt werden muss. Diese wird hier und im folgenden **Angebot** genannt. Ein Angebot muss alle möglichen Verkaufsarten (*Kauf, Miete, o.A.*) sowie dazu passende Preise beinhalten. Da die Preise und Verkaufsarten für Software je nach Automarke und -model variieren können sollen, wird ein Modul "pricing and sales", kurz PAS, zur Bestimmung des Preises und der Verkaufsart vorgeschlagen (*siehe Abbildung 5*). Das Director Repository ist nach wie vor dazu in der Lage, Softwareimages und Metadaten direkt an die Primäre Recheneinheit (ECU) zu schicken, wenn es sich nur um eine Aktualisierung bereits installierter Software handelt. Wird hingegen eine neue Software vorgeschlagen, so werden Image und Metadaten zusammen mit einem Angebot an den Software-User Pattern-Recognizer des Autos verschickt. Alternativ wird nur das Angebot verschickt um abzuwarten, ob der Kunde die Software tatsächlich haben will. Hierdurch würde das Image nicht unnötiger Weise verschickt werden, was unter anderem die Ressourcen schont. Der SUPR des Autos identifiziert unter stetiger Beobachtung des Fahrers, wann dieser einen Softwarevorschlag verarbeiten kann. Er stellt zum gegebenen Zeitpunkt Verbindung zu der Nutzeroberfläche des Wagens her, um so mit dem Fahrer zu kommunizieren und den Kaufvorgang abzuschließen (*Ob erfolgreich oder nicht ist hierbei egal*). Die Funktionsweisen und Aufgaben des SUPR im Auto unterscheiden sich von denen des SUPR im Director Repository. Aufgrund dessen ist es wichtig eine Abgrenzung und Spezifizierung dieser vorzunehmen.

### 2.3.3 SUPR im Rahmen der Softwarebereitstellung

Das folgende komplettiert den in Kapitel 2.3 vorgestellten Software-User Pattern Recognizer (*SUPR*). Wie im ersten Teil wird auch im Rahmen der Bereitstellungskomponente des SUPR zwischen den Aufgaben auf Server und denen im Fahrzeug unterschieden.

#### SUPR im Director Repository

Die unterschiedlichen Suchvarianten welche der SUPR im Rahmen der Bedarfserkennung bereits auf dem Server ausführt, wurden in Kapitel 2.3 spezifiziert. Im Rahmen der Software Bereitstellung ist es die wesentliche Aufgabe, das Angebot und alle möglichen Preise festzulegen. Die verschiedenen Preise ergeben sich aus Kombination der einzelnen **Einflussfaktoren**.

Ein Angebot, welches vom Director Repository (siehe 3.2) an das SUPR des Fahrzeugs geschickt wird, spezifiziert die Preise für Software in Abhängigkeit der **Kaufart**, der **voraussichtlichen Laufzeit** des Mietverhältnisses (bei Miete), der Menge der Software welche der Fahrzeughalter in dem Moment akquirieren möchte und den eigentlichen Herstellungskosten der Software. Die folgende Tabelle definiert die Werte, welche die einzelnen Einflussfaktoren annehmen können.

Wert	Beschreibung
<b>Kaufart</b>	
Kauf	Der Fahrer akquiriert die Software dauerhaft. Eine Reklamation ist möglich, wenn diese ihre Aufgabe nicht korrekt ausführen würde.
Leihe	Der Fahrer legt einen Zeitraum ( <i>von .. bis</i> ) fest, für welchen er diese Software akquirieren möchte. Für diesen bezahlt er im Voraus und die SW wird bei Ablauf des Zeitraums deinstalliert.
Miete/Abo	Der Fahrer wählt die Dauer einer Mietphase, an welche sich der Preis anpasst. Das Mietverhältnis wird nach Ablauf dieses Zeitraums aufrecht erhalten. Das heißt es entstehen laufend Kosten bis es vom Fahrer beendet wird.

Zeitraum/Dauer	
Permanent	Der Zeitraum ist nur dauerhaft, wenn die Software gekauft wird. Hierbei tritt der höchste Preis auf.
Lang	Sechs Monate oder mehr. Auf einen Tag heruntergerechnet der günstigste Miet- oder Leihpreis.
Mittel	Sechs Wochen bis zu sechs Monate. Auf einen Tag heruntergerechnet ein wenig teurer als der des Langen.
Kurz	Acht Tage bis sechs Wochen. Auf einen Tag heruntergerechnet ein wenig teurer als der des Mittleren.
Einmalig	Ein bis sieben Tage. Auf einen Tag heruntergerechnet das teuerste Angebot.

Menge	
Einzeln	Es wird nur eine Software zu dem Zeitpunkt angeboten. Hat keinen Einfluss auf den Preis.
Mehrere	Software wird im Bundle gekauft, was sich positiv auf den Preis auswirkt. Ein Bundle muss vom SUPR im Fahrzeug erstellt werden.
Flottenkauf	Es wird Software für mehrere Fahrzeuge gekauft. Dies kann für Unternehmen Sinnvoll sein, die Dienstwagen anbieten oder auch Autovermietungen etc.

Weitere Einflüsse auf den Preis	
Herstellungskosten	Die Kosten die im Laufe der (Weiter-)Entwicklung und der Wartung dieser Software entstanden sind.
Beliebtheit	Ist eine Software beliebt bzw. gefragt, steigt der Preis dieser. Die Beliebtheit kann entweder global gemessen werden oder regionsabhängig. Der Preisanstieg sollte dabei nicht zu hoch sein.
Mächtigkeit	Je mehr eine Software "kann", desto teurer sollte diese sein.

Nachdem der SUPR des Director Repositorys eine den Bedarf deckende Software gefunden hat, muss er die Referenz auf die Datei an das PAS-Modul (*siehe Abbildung*) schicken. Das PAS-Modul soll anhand der Software, einiger Informationen zum Kaufverhalten der Fahrer des Fahrzeugs sowie dem generellen Bedarf der Software ein personalisiertes *Angebot* erstellen und anschließend an das Fahrzeug schicken.

### SUPR im Fahrzeug

Neben dem Server-seitigen SUPR hat auch der SUPR des Autos wichtige Funktionen im Rahmen der Bereitstellung von Software. Ist das Angebot im Auto angekommen, identifiziert dieser einen geeigneten Verkaufszeitpunkt und stellt zu diesem das personalisierte Angebot über die Mensch-Maschine Schnittstelle (MMS) dar. Dies meint den Zeitpunkt, an dem der Verkauf oder die Leih von Software möglichst wahrscheinlich ist. Hierzu ist eine Beobachtung des Fahrers nötig, um anhand von dessen **Eigenschaften** wie bspw. Mimik, Gestik, Sprache oder dem Fahrverhalten das Stress-, Freude, oder Angstlevel zu deuten, aber auch um Müdigkeit oder Ablenkung feststellen zu können. Vor allem Faktoren wie Stress, Freude und Angst beeinflussen unsere Meinung und Entscheidung."Vgl. [30, S.44]. So wird eine Software eher nicht gekauft, wenn der Fahrer in eine stressige Verkehrssituation überblicken muss, zum Beispiel wenn es dicht benetzt ist und man zur Zeit in einer unbekannten Gegend Auto fährt. Hingegen beeinflusst Freude einen eher dazu, einen Kauf zu tätigen. Angst ist für den Anwendungsfall des Verteilens neuer Fahrfunktionen divers zu deuten. Wenn der Fahrer Angst vor der Strecke hat die vor ihm liegt, muss der SUPR dies anders bewerten als Angst die Aufgrund anderer Einflüsse entsteht. Diese Beobachtung entspricht der dritten Richtlinie Mensch-zentrierter autonomer Fahrzeuge nach Lex Fridman. [26, S. 3] Nach der fünften Richtlinie Fridmans, soll ein Auto von dem Moment dem ersten Einstiegs des Fahrers auf diesen personalisiert werden. [26, S. 5] So ist es Sinnvoll, würde der SUPR persönliche Eigenschaften in seine Entscheidungsfindung zum geeigneten Verkaufszeitpunkt einfließen lassen.

Ein System eines Fahrzeugs soll laut Fridman Daten-orientiert sein [26, S. 3]. Das heißt, dass ein Fahrzeug aus den aufgenommenen Daten lernen soll. Fusioniert man die bei der Fahrerbeobachtung aufgenommene Daten mit Daten über den Verkaufserfolg, kann der SUPR aus seinen Handlungen lernen und sich so optimieren. Hierdurch werden mögliche den Kauf negativ beeinflussende Fak-

toren identifiziert (zB. Stress) und der Fahrer so besser kennengelernt. Die Kaufbereitschaft des Fahrers soll in Folge dessen möglichst oft korrekt eingeschätzt werden.

Ist der Zeitpunkt bzw. Zeitrahmen identifiziert, stellt der SUPR über die Mensch-Maschinen-Schnittstelle das personalisierte Angebot dar. Der dargestellte Inhalt wird maßgeblich beeinflusst von den Aspekten der Verkaufspräzesspsychologie. Um die wesentliche Aufgabe des SUPR (*Beobachtung & Kommunikation mit Fahrer, Vermarkten von Software*) sinnvoll zu modellieren, werden die Einflussfaktoren für den dargestellten Inhalt der MMS anhand der Prinzipien der Verkaufspolitik nach Markus Reinke [28] erläutert.

Um einen Kunden vom Erwerb eines Produkts zu überzeugen, muss diesem etwas angeboten werden wodurch ein offenes Bedürfnis befriedigt. Dazu muss herausgefunden werden, welches Produkt geeignet ist, damit es so anschließend angeboten werden kann. Wie der SUPR die Bedürfnisse von Fahrern identifiziert ist in Kapitel 2.3 nachzulesen. Was bei dem Anbieten einer Software beachtet werden muss, wird in diesem Kapitel abgehandelt.

Kunden sind unterteilbar in Plus-, Chancen- und Minus-Kunden. [28, S. 10ff.] Minus-Kunden werden ein Produkt von Beginn an nicht kaufen wollen, so gut der Erwerb auch gestaltet sein wird. Plus-Kunden werden ein Produkt voraussichtlich kaufen. Um die Chancen-Kunden von einem Erwerb zu überzeugen, werden im Absatz von Unternehmen diverse Prinzipien angewendet. Es kann also die Schlussfolgerung gezogen werden, dass so gut der SUPR seinen Fahrer auch kennt und wie gut Produkte den Bedarf auch abdecken, der Kunde dennoch nicht immer eine Software erwerben wird. Um die Anzahl an Käufern möglichst hoch zu halten, soll der persönliche Nutzen den die SW für den Kunden hat dargestellt werden. So kann die Situation simuliert und dargestellt werden, in welcher das Fahrzeug den Softwarebedarf festgestellt hat.

### **Das Differenzprinzip** [28, S. 19fff.]

Nach dem Differenzprinzip sollen unterbewusste Reize gesetzt werden, die zum Erwerb leiten. Zum Beispiel wird mit der teuersten Variante des Produkts geworben, damit der Kunde bei näherem betrachten des Angebots die tieferen Preise, also ein "kleineres Übel", entdeckt und sich darüber erfreut. Dabei soll nicht die Frage **ob** der Kunde bei einem kaufen möchte, sondern **was** der Kunden von einem kaufen will.

Stellt der SUPR das Angebot dar, soll eine Mietfunktion im Fokus stehen, zudem deutlich gemacht werden. Die Option SSoftware Kaufen ist kleiner darzustellen zusammen mit dem jeweiligen Preis. Um dem Kunden die Möglichkeit zu lassen die Software nicht zu erwerben, muss ein dezentner Knopf in der Nutzeroberfläche sein, welcher den Angebotsprozess beendet. Durch die Unauffälligkeit kann der Kunde zum Kauf gelenkt werden bzw. zu der Frage **was** man kaufen möchte, nicht **ob**. Um einen unterbewussten Reiz zu setzen, kann eine Karte dargestellt werden wie häufig die Software in der Umgebung installiert wurde. Dies sollte allerdings nur geschehen, wenn die Nachfrage tatsächlich auffällig hoch ist. Hiermit würde auch der Nachahmungseffekt abgedeckt werden (*weiter unten*).

### **Das Do-ut-des-Prinzip** [28, S. 33fff.]

Nach diesem Prinzip, wird "gegeben, damit du gibst". Der Kunde wird durch beispielsweise Gratis-Proben angelockt um zum Erwerb bewegt zu werden. Will er dennoch nichts kaufen, kann um Weiterempfehlung gebeten werden. Dieser Bitte wird der Kunde vrs. nachkommen, da er im Vorfeld etwas "gratist erhalten hat. Die *SZwei Schritte vor, einer Zurück Taktik*" hierbei anzuwenden ist sinnvoll. Das heißt, dass dem Kunden zuerst die teuersten Produkte gezeigt werden, um ihn an-

schließend auch hier mit niedrigen Preisen anderer Produkte beeindrucken und halten zu können. Es sollte ist hierbei auch nicht das Ziel das teuerste Produkt zu verteilen, sondern das eigentliche Ziel ist eines mit tieferem Preis.

Der SUPR kann dieses Prinzip anwenden, in dem er dem Kunden zu neu erworbener Software eine Test-Version einer anderen Software schenkt", welche nach einem bestimmten Zeitraum (4 Wochen) abläuft. Diese Software sollte einen bestehenden Bedarf der Kunden abdecken.

### **Das Konsequenzprinzip**

Konsequenz im Handeln wird in den meisten Kulturen als eine positive Charaktereigenschaft gewertet, strahlt Sicherheit aus. Zudem schafft es Vertrauen bei Kunden. Konsequenz kann unter anderem mittels wenn-dann-Fragen ausgestrahlt werden, also "Wenn wir Ihnen XY bieten, kaufen Sie dann?" Der Kunde würde bei Erfüllung seines Bedarfs eher kaufen. Und da der von ihm benannte Bedarf abgedeckt ist, wird er so von einem frühen Commitment überzeugt. Es sollen die Wünsche und Prioritäten der Kunden erkannt werden und anschließend das Produkt mit Fokus auf diese beworben werden.

Dieses Prinzip sollte vor allem verwendet werden, um einen "Fuß in die Tür zu bekommen" nicht um das teure Produkt zu verkaufen. Sinnvoll ist es, nach dem Motto "Kleinvieh macht auch Mistfuß verkaufen. Der SUPR tut dies zum einen indem er die Situation, in welcher der Bedarf festgestellt wurde, auf der MMS darstellt und den Kunden so erinnert. Wenn der Kunde noch gar keine oder nur wenig Software erworben hat, soll ihm immer die Möglichkeit einer kostenlose Probe-Woche geboten werden. Dies soll bis zu fünfmal geschehen, danach wird ohne weiteres keine Gratis Software mehr angeboten.

### **Das Nutzen des Nachahmungseffekts [28, S. 67fff.]**

*"Der Mensch ist grundsätzlich ein Gemeinschaftswesen und achtet daher sehr darauf, was andere von ihm denken, sowohl in der breiten Öffentlichkeit als auch in seinem engen persönlichen Umfeld. Markus Reinke (Vgl. ) [28, S. 67]*

Dieses Verhalten, der sogenannte **Nachahmungseffekt**, kann durch das Einsetzen verschiedener Techniken provoziert werden. Bei der **Zeugenumlastung** lässt sich der Kunde bestätigen, dass ihr Unternehmen und Produkt "gut" ist indem er von anderen positives darüber in Erfahrung bringt. Eine Variante, wie diese vom SUPR angewendet werden kann, wurde zuvor bereits dargestellt. Bei der **Referenztechnik** werden die Stammkunden eines Unternehmens gebeten Empfehlungsschreiben zu erstellen mit welchen im folgenden geworben werden kann. Der SUPR kann hierzu ein Bewertungssystem einführen und Nutzer können die Bewertung anderer Fahrer lesen und sich so eine Meinung der Software einholen. Dabei ist es sinnvoll, einige vorgefertigte Antwort/Bewertungsmöglichkeiten selber zur Verfügung zu stellen. Eine dritte Technik ist das **Empfehlungsmarketing**, bei welchem Kunden gebeten werden uns an Freunde und Verwandte weiter zu empfehlen. In den Zeiten von Social Media bietet es sich an, eine "TeilenFunktion zur Verfügung zu stellen. Der Fahrer soll Teilen können, wie viele Kilometer er in welcher Zeit zurückgelegt hat, wie viele davon autonom bewältigt wurden und welche Softwarepakete genutzt wurden.

Um eine hohe Resonanz herzustellen, ist es sinnvoll dem Kunden ähnlich zu sein, sympathisch zu wirken, Lob & Anerkennung zu verteilen als auch andere zu unterstützen. Da es ungewohnt ist, Lob oder anderes von einem Computer zu erhalten, der SUPR aber dennoch eine hohe Resonanz erzielen soll, sollte den Kunden eine Bezugsstelle gegeben werden. Eine Referenz aus der

Wirtschaft hierzu stellt "Meet Olli" dar [2]. Olli ist das Maskottchen des UserInterfaces (UI) von "Local Motors" (LM). LM stellt einen autonom fahrenden Bus her, mit welchem die Insassen über eine UI interagieren können. Olli hat "nur" zwei Augen und einen Mund, stellt hierdurch aber einen vermenschlichten Bezugspunkt für den Fahrer und die Insassen dar und gewinnt so das Vertrauen der Insassen.

Darüber hinaus soll der Kunde nicht von den Angeboten "erschlagen" werden, sondern der SUPR muss diese dosiert vorlegen. Wenn der Kunde kein Interesse hat, ist dies zu respektieren und er soll damit nicht weiter konfrontiert werden. Der SUPR soll nur die tatsächlich benötigte Software bewerben - ein einmalig aufgetretener Bedarf fällt da nicht drunter (*Die Ausnahme ist, der Kunde ist ein absoluter Plus-Kunde*). Es muss genügend Werbung betrieben werden damit möglichst viel Software verkauft wird. "Klassische" Werbung auf der MMS ist nicht Sinnvoll, sondern eher Werbung in Form von Gratis-Proben von Software-Paketen. Nutzt der Fahrer eine dieser Proben und die SW kommt zum Einsatz, kann der SUPR den Fahrer über die MMS darüber in Kenntnis setzen, wodurch er vor Augen geführt bekommt, dass er diese Software braucht.

Ein Auto wird allerdings nicht nur von einer Person gefahren, sondern möglicherweise von mehreren Familienmitgliedern, Mitarbeitern oder anderen Mietern im Falle von Carsharing. Das SUPR muss dazu in der Lage sein, diese einzelnen Personen auseinanderhalten zu können um keine falschen Entscheidungen zu treffen.

Wird dem SUPR ein Softwarevorschlag mit Angebot zugeschickt, soll dieser bis das Angebot angezeigt werden kann, diese Software schon anfangen herunterzuladen. Dadurch kann die Software, wenn sie erworben wird, bereits während der Fahrt installiert werden, was den Verkauf schneller und Nutzerfreundlicher gestaltet.

#### 2.3.4 Die Mensch-Maschine-Schnittstelle

Die Mensch-Maschine-Schnittstelle ist das System, über welches der Fahrer und das Auto mit einander interagieren. Im Kontext intelligenter Fahrzeuge handelt es sich hierbei meist um einen Bildschirm, mitunter haben Fahrzeuge auch einen integrierten Sprachassistenten.

Das Bereitstellen neuer Fahrfunktionen schließt mit ein, dass ein Fahrzeug zum Zeitpunkt A selbstständig Fahren kann und zum Zeitpunkt B nicht. Zwischen diesen Zeitpunkten muss eine Übergabe der Fahraufgabe von dem Fahrzeug an den Fahrer erfolgen. Dabei soll der Fahrer des Fahrzeugs über die momentane Verkehrslage ausreichend in Kenntnis gesetzt werden, um während und nach der Übergabe der Fahraufgabe die Sicherheit zu wahren. Der Begriff ausreichend definiert sich dabei wie folgt:

**Die Inkenntnissetzung gilt als vollständig, wenn die Mensch-Maschine-Schnittstelle alle Prinzipien von S. Debernard et Al. [29] erfüllt.**

Das zu entwickelnde Display soll Nutzer-zentriert (User-Centered) entwickelt werden, um die Anforderungen möglichst vieler Nutzer erfüllen können und zusätzlich den Faktor der "Vermenschlichung" zu erhöhen. Die geschaffene transparente Straßenführung über das Display eines Fahrzeugs, soll zusätzlich durch Audio-Assistenten unterstützt werden. Die MMS wird so durch ein audiовisuellem System.

Die Schnittstelle soll sich sowohl visuell als auch auditiv an die im Fahrzeug sitzenden Personen

anpassen. So kann zum Beispiel die Schriftgröße größer sein, wenn eine Person im Rentenalter das Fahrzeug betritt oder ähnliches. Mögliche Features der Personalisierung (*z.B. größere Icons, dunkles Design*) werden mit Hilfe der Personas 2.4.1 in der Bachelorarbeit erarbeitet. Im generellen soll der Nutzer auf dem Display eine Simulation des eigenen Autos sehen zusammen mit zusätzlicher Information gemäß den Prinzipien nach Debernard et. Al. Das Design des User-Interfaces kann vom Fahrer angepasst werden, wobei die intuitive Bedienung sich nicht verschlechtern soll. Durch die Vermenschlichung des MMS, einer Transparenten Straßenführung, der Möglichkeit die Steuerung jederzeit zu übernehmen sowie der höflichen Kommunikation mit dem Fahrer, wird das Vertrauen des Fahrers zu dem Fahrzeug gefördert [2], was den Kauf von Software wahrscheinlicher macht.

## 2.4 Technologie- und Methodik-Scouting

Um auf Grundlage der dargestellten Rahmungen für die Bedarfserkennung und Bereitstellung neuer Fahrumfänge für intelligente Fahrzeuge einen Prototypen zu entwickeln, wird abschließend ein Technologie- und ein Methodik-Scouting durchgeführt werden. Das Kapitel der Arbeitsmethodik spezifiziert Den Zyklus des User-Centered-Designs und stellt dar, weshalb dieser für die Entwicklung des Prototypen die richtige ist. Das anschließende Technologie-Scouting stellt Entwicklungsumgebungen, Frameworks und Programmiersprachen vor, welche in der Entwicklung genutzt werden.

### 2.4.1 Arbeitsmethodik: User-Centered-Design

Um die Nutzeroberfläche für die Zielgruppen ansprechend zu gestalten, wird bei der Erarbeitung dieser nach den Prinzipien des User-Centered Designs (*UCD*) gehandelt. Ins deutsche übersetzt bedeutet dies "Benutzerzentriertes Design bzw. Benutzerorientierte Gestaltung eines Produkts. Es beschreibt einen Designprozess bzw. ein Entwicklungsverfahren, auf welches der Endnutzer (Fahrer) schon von Anfang an Einfluss nimmt" (Vgl. [24, S.763]). Der User wird in die Phasen der Entstehung einer Benutzungsschnittstelle integriert was zur Folge hat, "dass der Aufbau, die Inhalte und deren Form sowie das Design des Endproduktes maßgeblich von den Bedürfnissen, Erwartungen und dem Verständnis der User bestimmt wird" (Vgl. [1]). Des weiteren kann durch das Einbeziehen von Kunden die Qualität optimiert werden (Vgl. [12, S. 14]).

"Typisch für einen UCD-Prozess zum Entwerfen von Webanwendungen mit optimaler User Experience sind [jedoch] die Prozessschritte Analyse, Konzeption, Umsetzung/ Design, Evaluierung und Optimierung."(Vgl. [11])

Im Rahmen der Analyse werden die Anforderungen an das System erstellt und zusammengeführt. Hierzu können Personas erstellt werden (siehe: 2.4.1) und aus eben diesen können Anforderungen erstellt werden. Die Analyse soll zudem "Usability" [14] Ziele festlegen, anhand welcher in der "Evaluation und Optimierung" die Güte der Nutzeroberfläche gemessen werden kann [11]. Während der Konzeption soll das Verständnis der Benutzer und deren Bedürfnisse hinsichtlich der User Experience auf die Benutzeroberfläche übertragen werden [11, (ebd.)]. Die Konzeption endet mit dem erstellen eines Prototypen. Dieser wird in der Dritten Phase (Design) durch ein konsistentes, ansprechendes und klares Grafik Design" (Vgl. [11]) erweitert, was Probleme löst und eine Intuitive Bedienung unterstützt. Die Phase der "Evaluation und Optimierung" wird das Produkt auf Probleme wie Sicherheit oder eine schlechte Bedienbarkeit hin untersucht. Hierdurch werden Mängel entdeckt welche in den nächsten Zyklus der Entwicklung einfließen.

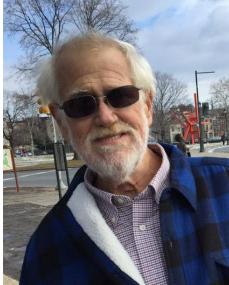
Wie hoch der Grad der Usability und der User-Experience ist, lässt sich wie zuvor erwähnt anhand von Personas ableiten. Im Folgenden wird daher erklärt, was Personas sind und es werden eigene Personas für das Projekt erstellt.

### Personas

Bei Personas handelt es sich nun um fiktive Personen, also hypothetische User, mit individuellen Eigenschaften [9, ], welche ebenso eine reale Person haben könnte. Je ähnlicher die Personas also der Zielgruppen des Unternehmens/des Produktes sind, desto effektiver ist die Verwendung dieser. Personas sind nicht nur als potentielle Zielgruppe zu sehen - sie haben zudem die Aufgabe, dass sich das Entwicklerteam in die Position des Nutzers versetzen kann. Dazu werden Personas zunächst nach Kategorien wie "Geschlecht/Alter" aufgeteilt und anschließend wird jedes Profil mit Merkmalen und Eigenschaften gefüllt. Welche Informationen unter anderem zu Personas hinzugefügt werden, zeigt die folgende Tabelle.

Vor- und Nachname	Geburtsdatum/Alter
Foto der Person/Aussehen	Herkunft/Wohnort
Sprachkenntnisse	Beruf; Berufserfahrung in Jahren
Bildung/Ausbildung	Familienstand
Interessen und Hobbys	Fähigkeiten/Behinderungen
Sicherheitsrisikofaktoren für den Straßenverkehr	Abneigungen
Erfahrungen mit Technik (Meidenkompetenz)	Vorlieben
Fahrerfahrung	Kaufbereitschaft
Auto; Wenn ja: Besitzverhältnis?	

Anhand dieser möglichen Informationen werden im folgenden die Personas für dieses Projekt erstellt.

Name / Soziographische Daten	Hobbys & Interessen	Anwendungsfallbezogenes
 <ul style="list-style-type: none"> <li>• Dietmar Müller, 68 Jahre (Bild: [10])</li> <li>• Loppersum, Ostfriesland</li> <li>• Deutsch</li> <li>• Gelernter Maurer</li> <li>• Verheiratet, 3 Erwachsene Kinder</li> </ul>	<ul style="list-style-type: none"> <li>• Rentner; Taxiunternehmer (seit 20 Jahren aktiv)</li> <li>• Fischen, Wandern</li> <li>• Handwerklich begabt, Hobbygärtner</li> <li>• Diabetiker</li> <li>• idR. Minus-Kunde</li> <li>• Geizig</li> </ul>	<ul style="list-style-type: none"> <li>• Führerschein seit 47 Jahren</li> <li>• Vertraut Technik nicht</li> <li>• Schlechte Sehkraft</li> <li>• Ängstlicher Autofahrer</li> <li>• Besitzt eigenen Neuwagen</li> </ul>
 <ul style="list-style-type: none"> <li>• Jake Schneiders, 52 Jahre (Bild: [3])</li> <li>• Washington, USA</li> <li>• Englisch, Russisch (Fließend)</li> <li>• Ausgebildeter Cyberhacker</li> <li>• Geschieden, ein Kind (geteiltes Sorgerecht)</li> </ul>	<ul style="list-style-type: none"> <li>• Leutnant beim Militär</li> <li>• Jagen, Baseball Trainer</li> <li>• Kindersorgerecht unter der Woche</li> <li>• Workaholic</li> <li>• Chancen-Kunde</li> </ul>	<ul style="list-style-type: none"> <li>• Führerschein seit 27 Jahren</li> <li>• Vertraut sehr auf Technik</li> <li>• guter, sicherer Autofahrer</li> <li>• Setzt Wert auf gute Bedienbarkeit</li> <li>• Hat gerne die Kontrolle über Systeme</li> <li>• Fährt einen Wagen des Militärs</li> <li>• Arbeitet viel am Laptop</li> </ul>

 <ul style="list-style-type: none"> <li>• Chi Nguyen, 24 Jahre (Bild: [13])</li> <li>• Rom, Italien</li> <li>• Vietnamesisch, Italienisch(Fließend), Englisch (Fließend)</li> <li>• Studentin (Geschichte)</li> <li>• Single</li> </ul>	<ul style="list-style-type: none"> <li>• Studentin, Stadtführerin in Rom</li> <li>• Joggen, Roadtrips</li> <li>• Influencer</li> <li>• Plus-Kundin</li> </ul>	<ul style="list-style-type: none"> <li>• Führerschein seit 2 Jahren</li> <li>• Vertraut sehr auf Technik</li> <li>• unsichere Autofahrerin</li> <li>• Ist mit viel Technik aufgewachsen</li> <li>• Mag es, ihre Apps zu personalisieren</li> <li>• Nimmt Teil an Car-Sharing</li> </ul>
--	---	---

Die erstellten Personas werden bei der Erstellung des Prototypen herangezogen um Anforderungen an das System. Jede einzelne ermöglicht es, bestimmte Schwierigkeiten und Herausforderungen genauer darzustellen. So soll Dietmar Müller verdeutlichen, dass ein Kunde möglicherweise doch vom Kauf einer SW überzeugt werden kann, obwohl er idR. ein Minus-Kunde ist. Durch ihn kann auch dargestellt werden, welche weiteren Einflussfaktoren es neben Stressö.Ä. gibt, wie bei seine Diabetes Erkrankung.

Jake Schneiders soll die Art Person darstellen, welche neugierig ist, die Sicherheit eines System testen möchte und hierzu auch selber in der Lage ist. Durch seine technische Affinität kann er Systeme leicht Verstehen und neue Anforderungen erstellen, die voraussichtlich leicht zu implementieren sind.

Chi Nguyen ist eine sehr feminine Studentin welche bereits vielerorts gewohnt hat. Durch ihr Dasein als Influencerin hat sie ein großes Netzwerk an Followern, mit welchen sie ihre Erfahrungen die sie während der Autofahrt sammelt Teilen kann.

Da alle Personas in unterschiedlichen Ländern an dem Straßenverkehr teilnehmen, sind die Einflussfaktoren zur Erstellung von Anforderungen sehr vielfältig, was sich positiv auf die Entwicklung auswirken kann.

#### 2.4.2 Tech-Scouting

Damit die Entwicklung des Prototypen möglichst simpel sein wird, wird im Folgenden ein Technologie-Scouting durchgeführt. Es soll die Vorteile aufzeigen, welche ausgewählte Entwicklungsumgebungen, Programmiersprachen und Frameworks für die Entwicklung des Prototypen haben.

Im Rahmen eines Prototypen für die Erkennung und Bereitstellung neuer Fahrfunktionen für intelligente Fahrzeuge ist es nötig, ein auf der Straße fahrendes Fahrzeug zu Simulieren und dazu

eine passende Nutzeroberfläche zu haben, welche als Mensch-Maschine Schnittstelle des Fahrzeugs agiert. Für das Simulieren einer Straßenverkehrssituation wird der OpenSource Simulator **Carla**<sup>7</sup> verwendet. Mit Carla ist es möglich, das eigene Auto zusammen mit anderen Fahrzeugen, Radfahrern, Fußgängern, Hindernissen uvm. auf der Straße darzustellen. Autos können mit Sensoren ausgestattet werden und die aufgenommenen Daten können mittels der Python-API ausgelesen werden. Der Ablauf einer Simulation wird in einem Python-Script spezifiziert. Eine alternative Möglichkeit zur Erstellung einer Simulation ist das einbinden einer OpenSCENARIO-Datei. Es existiert ein Aufsatz auf Carla, wodurch der Simulator den in einer solchen Datei dargestellten Ablauf selber darstellen kann.

Für die Erstellung von Python Skripts wird "**PyCharm**"<sup>8</sup> als Entwicklungsumgebung gewählt. PyCharm ermöglicht eine intuitive und einfache Anbindung an GIT. Da die aus Carla ausgelesenen Daten verarbeitet werden müssen um zu erkennen wann das Auto nicht mehr selbstständig fahren kann, ist die Entwicklung eines Servers notwendig. Neben der Analyse der Simulation soll dieser als Kommunikationszentrale zwischen eben diesem und der Mensch-Maschine-Schnittstelle dienen. Zur Entwicklung des Servers soll das ebenfalls von JetBrains entwickelte **IntelliJ IDEA**<sup>9</sup> dienen. Auch IntelliJ bietet eine einfache Anbindung an GIT an - zusätzlich ist die Einbindbarkeit von **Maven**<sup>10</sup> zum verwalten von *Dependencies* ein leichtes.

Für die Entwicklung der Mensch-Maschine-Schnittstelle wird eine Android App entwickelt - für diese wird **Android Studio**<sup>11</sup> verwendet. Android Studio ermöglicht neben der visuellen auch eine textuelle Bearbeitung von Nutzeroberflächen. Es ist die von Google empfohlene Entwicklungsumgebung und ermöglicht das erstellen von Apps massiv.

Neben Python wird zum entwickeln von Server und MMS hauptsächlich die Programmiersprache Java (Version 1.10) verwendet. Java wurde an der Universität Oldenburg gelehrt und es wird unter anderem zum entwickeln von Android Apps verwendet.

## 2.5 Ausblick und erstes Konzept der praktischen Umsetzung

Nachdem in den vorherigen Kapiteln einzelne Aspekte der Wertschöpfungskette erläutert wurden, soll es abschließend einen Ausblick auf die Bachelorarbeit geben und ein erstes technisches Konzept des Prototypen vorgestellt werden. Im Laufe der Arbeit haben sich einige zu erläuternde Aspekte aufgetan, welche aus Platzgründen erst in der Bachelorarbeit behandelt werden.

Im Rahmen dieser soll zunächst ein erster Zyklus des UCD durchgeführt werden, um so die Nutzeroberfläche der Mensch-Maschinen-Schnittstelle zu erarbeiten. Dabei sollen zusätzlich mögliche Personalisierungsfeature benannt werden. Weiterhin sollen die anderen im Folgenden vorgestellten Teilsysteme entwickelt werden. Unter anderem soll ein Suchalgorithmus für den SUPR des Servers erarbeitet werden sowie eine Gegenüberstellung von Rechenleistung und Sendeleistung des Autos bzw. des Servers erfolgen. Es werden die einzelnen Bausteine einer Wertschöpfungskette (zur Erkennung und Bereitstellung neuer Fahrumfänge intelligenter Fahrzeug) benannt, erläutert und im Prototypen veranschaulicht. Im Folgenden wird für diesen ein erstes technisches Konzept vorgestellt.

---

<sup>7</sup><http://carla.org/>

<sup>8</sup><https://www.jetbrains.com/de-de/pycharm/>

<sup>9</sup><https://www.jetbrains.com/de-de/idea/>

<sup>10</sup><https://maven.apache.org/>

<sup>11</sup><https://developer.android.com/studio>

### **Erstes technisches Konzept des Prototypen**

Abbildung 6 zeigt eine erste Architektur des Prototypen. Dieser ist unterteilt in das **Fahrzeug**(unten) und den **Server**(oben). Das Fahrzeug ist in drei Teilsysteme aufgeteilt. Der "Carla Python Skript Client" ist das Modul in dem die Simulation stattfindet. Entscheidungen bzgl. der Fahraufgabe werden in diesem getroffen. Die **Mensch-Maschine-Schnittstelle** ist eine Android-basierte App. Sie wird über einen Emulator bedienbar sein und soll unter anderem Informationen aus dem Carla Simulator darstellen. Damit dies möglich ist, wird der dritte Baustein des Fahrzeugs eingeführt: der SUPR.

Der SUPR des Fahrzeugs soll die im Forschungsseminar definierten Aufgaben durchführen können und dient zusätzlich als Kommunikationseinheit. Er verbindet die Carla Simulation und die Mensch-Maschine-Schnittstelle, sodass eine interne Kommunikation im Fahrzeug möglich ist. Desweiteren dient der SUPR als Kommunikationseinheit zum Server. Er ist für die Installation eingehender Softwarevorschläge zuständig und muss entsprechende Updates an den Python Client senden. In der Uptane Architektur waren hier die ECU's eines Autos angedacht. Da diese in dem ersten Konzept nicht auftauchen, steuert dies der SUPR.

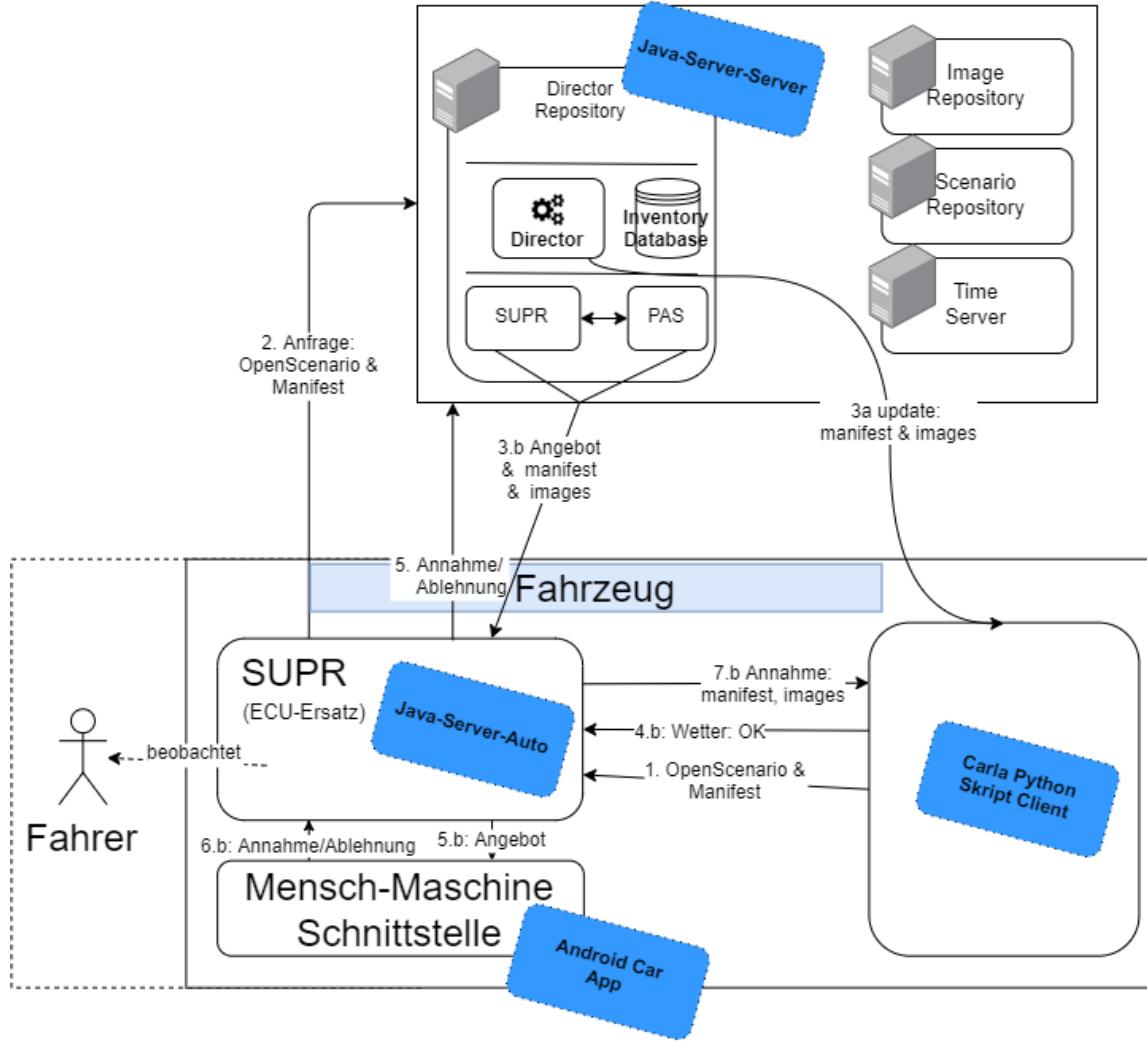


Abbildung 6: Technisches Erstkonzept

Die Architektur des Servers ist identisch zu der aus Kapitel 3.2. In der Implementierung sollen der Einfachheit halber die vier Bestandteile des Servers auf einem Server laufen und nicht getrennt werden. Der SUPR des Director Repositorys soll die im Forschungsseminar ihm zugeschriebenen Aufgaben durchführen können, gleiches gilt für den PAS. Die einzelnen Bestandteile des Servers können mit Ausnahme des Time-Servers miteinander kommunizieren.

Aus den Pfeilen des Konzepts abzuleiten ist der mögliche Ablauf eines Kaufprozesses. Initiiert wird dieser dadurch, dass das Fahrzeug im Carla Client nicht mehr autonom fahren kann. Dieser erstellt infolgedessen eine OpenScenario-Datei und schickt diese gemeinsam mit dem Manifest, in welchem die bereits installierte Software notiert ist, an den SUPR (Pfeil 1). Dieser schickt den entdeckten Bedarf als Anfrage an den Server (Pfeil 2), welcher infolgedessen überprüft welche Software den entdeckten Bedarf abdecken kann. Hierzu sucht der Director des Servers im Manifest des Fahrzeugs nach zu aktualisierender Software. Wird ein Defizit festgestellt, stellt der Server eine Verbindung mit dem Python Client her, welcher dieses Update installiert (Pfeil 3.a). Ist jede Software wieder aktuell stellt der SUPR des Servers fest, ob die Bedarfssituation mit den Updates bewältigt werden kann. Ist dies **nicht** der Fall, sucht der SUPR nach der passenden Software anhand der Open-

Scenario Datei. Ist ein Paket gefunden, wird das vom PAS erstellt Angebot zusammen mit dem unterschriebenen Manifest und dem Software-Image an den SUPR zurückgeschickt (Pfeil 3b).

Wird eine neue Software vorgeschlagen, hält der SUPR den Vorschlag so lange von der Nutzeroberfläche fern, wie der Fahrer sich noch in einer 'stressigen' Situation befindet. Ob dies der Fall ist, wird anhand des Wetters abgeleitet. Ist es regnerisch, nebelig oder ähnliches, soll das Angebot zurückgehalten werden. Bei sonnigem Wetter darf es dann auf der Nutzeroberfläche angezeigt werden. (Pfeil 4.b) Der Fahrer wird anhand einer Mitteilung darüber informiert, dass eine neue mögliche Software vorhanden ist. Der Fahrer kann das Angebot in der Nutzeroberfläche anpassen und es abschließend entweder annehmen oder ablehnen (Pfeil 6.b). Wird das Angebot angenommen, installiert der SUPR (Java-Client) die Software auf dem Fahrzeug, indem er eine Nachricht an das Python Skript schickt. Die Simulation wird erneut gestartet und das Auto kann die zuvor schwierige Situation bewältigen.

*Hier endet das Forschungsseminar.*

## 2.6 Eingrenzung des Themas

Nach der Präsentation der Ergebnisse des Forschungsseminars am 7. Februar 2020 in Braunschweig wurde in Absprache mit den Betreuern der Umfang der Bachelorarbeit eingegrenzt. Die Aspekte des User-Centered-Designs werden im Kontext der Arbeit nicht weiter beachtet, statt dessen soll ein stärkerer Fokus auf die Bausteine der Wertschöpfungskette und den zu entwickelnden Prototypen gelegt werden, um die Bedarfserkennung und Bereitstellung von Software möglichst detailliert darstellen zu können.

Des weiteren wurde sich darauf geeinigt, den im Prototypen dargestellten Anwendungsfall zu ändern. Zuvor war vorgesehen, einem Fahrzeug anhand der im Forschungsseminar skizzierten Suchalgorithmen eine neue Software vorzuschlagen. In dem neuen Anwendungsfall wird ein Fahrzeug eine Software installieren und nutzen können, welche es dem Fahrzeug ermöglicht selbstständig auf einem Parkplatz parken zu können. Die Ergebnisse des Forschungsseminars, insbesondere die skizzierten Suchalgorithmen und die Integration von OpenScenario-Dateien werden daher **nicht** im Prototypen implementiert, finden aber dennoch in der Bachelorarbeit Einfluss. Durch die Änderung des Anwendungsfalls konnten weitere Teilnehmer der Supply Chain identifiziert und somit ein umfassender Überblick des Marktes für die Erkennung und Bereitstellung neuer Fahrumfänge erstellt werden.

### 3 Das Business Model Canvas und die Wertschöpfungskette

Damit die Verteilung von **Software** geeignet organisiert wird, bedarf es einer Plattform über welche diese orts- und zeitunabhängig heruntergeladen werden können. Diese Plattform kann in Form eines Software Shops realisiert werden, über welchen Softwares von Fahrzeughaltern gekauft und installiert werden können. Ein passender Vergleichshaus der Realität ist der Google Play Store. Der Shop stellt den Mittelsmann zwischen Fahrzeughaltern und den Softwareherstellern dar. Die installierten Softwares können unter anderem den Fahrumfang autonomer Fahrfunktionen erweitern oder das Auto mit anderen Akteuren des Straßenverkehrs verbinden. Ein bereitgestellter Shop sollte Automarkenübergreifenden entwickelt werden, da hierdurch zugleich ein großer Teil des Marktes gewonnen wird sowie einige Mehrwerte für Fahrzeughalter geschaffen werden wie zum Beispiel

#### 1. Nachhaltigkeit von Fahrzeugen

Nach dem verlassen des Fließbandes altert die Software eines Autos. Updates sind heutzutage nur beim Mechaniker möglich und sind daher mit einem großen Zeitaufwand für Fahrzeughalter verbunden. Mittels einer Kabelfreien Schnittstelle können Fahrzeughalter gewünschte Software orts- und zeitunabhängig installieren. Bleibt die Software von Fahrzeuge länger „*aktuell*“, kann die Nachfrage an Neuwagen langfristig zurückgehen und sich der Automobilmarkt so grundlegend ändern. Ist der Shop Automarken-übergreifend, können sämtliche Softwares des Shops für jeden Fahrzeughalter weltweit verfügbar sein.

#### 2. Selbstständige Erweiterung des Fahrzeugs

Die schlechte Alternative zu einer kabelfreien Bereitstellung von Software ist die stetige Fahrt zum Mechaniker. Hier erhält ein Fahrzeug eine festgelegte Sammlung an Softwares, der Fahrzeughalter hat also keine Auswahlmöglichkeiten.<sup>12</sup> Auf Dauer kann daher viel Software auf dem Fahrzeug installiert sein, die der Fahrzeughalter nicht benötigt. Durch einen Softwareshop wird es möglich, dass ein Fahrzeughalter nur die tatsächlich benötigte Software auf seinem Fahrzeug installiert hat. Des Weiteren können die Kosten für Fahrzeughalter hierdurch skalierbar gehalten werden, da diese selber entscheiden welche Softwares gekauft werden.

Um einen Überblick des Marktes zu geben, wird zunächst ein Business Model Canvas (*BMS*) erarbeitet. Anschließend werden relevante Bausteine der Wertschöpfungskette anhand der Erkenntnisse aus dem Forschungsseminar sowie des Business Models identifiziert und deren Aufgaben erläutert.

#### 3.1 Business Model Canvas

Das 2004 von Alexander Osterwalder entwickelte Business Model Canvas (*BMC*) schafft einen Überblick über die Aufgaben, die Kosten- und Partnerstrukturen sowie den Kundensegmenten eines Unternehmens. Es hilft den Fokus auf die wesentlichen Zielsetzungen dessen zu setzen. [16, Vgl. ] Folgend werden die Kundensegmente, die Kundenbeziehungen, die Marketingkanäle und die Einnahmequellen betrachtet anhand welcher anschließend die Nutzenversprechen sowie die Schlüsselressourcen und -aktivitäten eines Shops bestimmt werden. Durch die abschließende Bestimmung von Schlüsselpartnern und der möglichen Kostenstruktur eines Softwareshops wurden „alle wesentlichen Elemente eines Geschäftsmodells in ein skalierbares System gebracht“ [20, S. 14], anhand wessen

---

<sup>12</sup>quelle

die Bausteine der Wertschöpfungskette abgeleitet werden können.

### 3.1.1 Kundensegmente

Die Kundensegmente eines Shops lassen sich in Einzelkunden und Flottenbetreiber unterteilen, wobei sich Flottenbetreiber in zwei weitere Segmente aufteilen lassen: "*Leihe und Leasing*" umfasst Autovermieter, Unternehmen die ihren Mitarbeitern Leasingwagen bereitstellen, aber auch weitere wie Car-Sharing Unternehmen. Deren Kunden und Mitarbeiter erwarten eine grundlegende Sammlung an Software im Mietfahrzeug vorzufinden und wollen möglicherweise selbstständig weitere Softwares auf eigene Rechnung installieren können. Neben „Leihe und Leasing“ sind auch Unternehmen mit Firmenwagen wie zum Beispiel Lieferdienste, Taxi-Unternehmer oder Bauunternehmen ein gesondertes Kundensegment. Beide haben kleine oder große Fahrzeugflotten und müssen Softwarekäufe dementsprechend skalieren können.

Einzelkunden sind die übliche Autofahrer, die ein privates Fahrzeug besitzen und Software auf diesem installieren möchten. Durch die enorme Größe ist es sinnvoll dieses Segment weiter zu unterteilen.

Sowohl Einzelkunden als auch Flottenbetreiber haben ähnliche Anforderungen an einen Software Shop. Durch neue Softwares soll ein Fahrzeug vermehrt selbstständig fahren können und den Insassen so Zeit zu sparen. Kunden sollten akquirierte Softwares verwalten und überwachen können, um so einen Überblick ihres Fahrzeugs zu haben.

### 3.1.2 Kundenbeziehungen

Um die Kunden nach dem ersten Kauf nicht zu Verlieren, muss eine positive Bindung zwischen ihnen und dem Shop aufgebaut werden. So sollte die erste Software, die dem Kunden vorgeschlagen wird einen deutlichen Mehrwert für diesen bieten. Hierdurch steigt die Zufriedenheit des Kunden und ein erneuter Kauf ist wahrscheinlicher.

Um langfristig viel Software über den Shop absetzen zu können, ist auch darüber hinaus eine gute Kundenbeziehung wichtig. Fahrzeughalter müssen **dem Shop vertrauen** können und bei **Kaufentscheidungen unterstützt** und **beraten** werden. Das Vertrauen kann gesteigert werden indem akquirierte Softwares deutliche Mehrwerte für Fahrzeughalter bieten. Auch die Einbeziehung von Fahrzeughaltern in die Entwicklung von Softwares kann eine positive Kundenbeziehung fördern. Um Flottenbetreiber beim Kauf von Software zu unterstützen, ist eine Web-App Sinnvoll, über die für eine große Menge an Fahrzeugen Softwares gekauft, verwaltet und überwacht werden können.

### 3.1.3 Marketingkanäle

Es ist davon auszugehen, dass ab dem Zeitpunkt des Markteintritts hergestellte Fahrzeuge von Partnerunternehmen den Shop vorinstalliert haben. Fahrzeughalter können daher umgehend neue Softwares kaufen und sollte über diesen Umstand möglichst häufig informiert werden. Hierzu sollten Werbekampagnen auf so vielen Marketingkanälen wie möglichen erfolgen, damit der Shop und die durch ihn geschaffenen Mehrwerte allgegenwärtig in der Gesellschaft werden. Zu Anfang könnten Rabatte oder weitere Sonderaktionen neue Kunden anlocken und vom Kauf überzeugen. In Zeiten von Social Media ist das Influencer Marketing wichtig, da diese Plattformen alleine in Deutschland täglich für ca. 140 Minuten genutzt werden. [17, Vgl. ]

### **3.1.4 Nutzenversprechen**

Nutzenversprechen sind die Wertever sprechen eines Unternehmens, die bestimmte Bedürfnisse von Kunden abdecken. Sie können von Personas, Interviews, Umfragen oder anderen abgeleitet werden.

#### **1. Orts- und Zeitunabhängige Akquise eigens ausgewählter Softwares**

Softwares sollen von Fahrzeughaltern orts- und zeitunabhängig heruntergeladen werden können. Die Softwares können selber ausgewählt werden, wodurch nur tatsächlich benötigte Softwares installiert und einige Fahrten zum Mechaniker verhindert werden.

#### **2. Überwachung, Verwaltung und Vorschlagen von Software**

Fahrzeughalter sollen überwachen können, welche installierten Softwares vom Fahrzeug wie oft genutzt werden. Hierdurch können nicht benötigte Softwares identifiziert, deinstalliert und anschließend möglicherweise im Shop bewertet werden. Neben dieser Unterstützung, wird auch die Suche nach möglicherweise passenden Softwares für den Fahrzeughalter automatisiert. Dies kann vor allem weniger technikaffine Kundensegmente beim Kauf unterstützen und Sicherheit im Umgang mit dem Fahrzeug schaffen. Wie der Bedarf einer Software erkannt werden kann, wird in den Kapiteln 2.2.3 und 4.2 erläutert.

#### **3. Autofahren wird sicherer**

Autonom fahrende Fahrzeuge könnten den Straßenverkehr sicherer zu machen, da mittels Sensoren und Recheneinheiten ein Fahrzeug mehr Daten aufnehmen und verarbeiten kann als der Mensch. [19, Vgl.] Neben den Sicherheitsvorteilen die einzelne Softwares für ein Fahrzeugs haben können, ist weitblickend vor allem der Einsatz von V2X-Kommunikation wertvoll, da diese die Sicherheit des Straßenverkehrs entscheidend verbessern kann werden. [6, Vgl.] [25, Vgl. S. 19]

#### **4. Lebenszeit des Autos wird verlängert**

Aufgrund der steigenden Sicherheit ist es wahrscheinlich, dass Fahrzeuge künftig unfallfreier Fahren können<sup>13</sup>. Durch den stetigen Kauf neuer und geregelten Updates bereits gekaufter Softwares bleiben die Softwares von Fahrzeugen länger aktuell und können so die durchschnittliche Lebenszeit eines Fahrzeugs verlängern, was die Anschaffung eines neuen Fahrzeugs aufschieben kann.

#### **5. Geringerer Wertverlust von Fahrzeugen**

Durch das eigenständige erweitern des Fahrumfangs der Fahrzeugs kann der Wert steigen. Nicht nur steigert Preis der Software den Wert, sondern auch die bereits vorhandene Konfiguration der Softwares beinhaltet einen Wert an sich.

#### **6. Fahrzeughalter gewinnen Zeit**

Durch die Integration neuer Fahrfunktionen können zunehmend mehr Situationen des Straßenverkehrs zurückgelegt werden ohne dass der Fahrer selber die Steuerung übernehmen muss. Die hierdurch gewonnene Zeit kann von den Insassen zum Arbeiten, gemeinsamen Interaktionen oder anderweitig genutzt werden. Durch diese gewonnene Zeit kann eine Autofahrt künftig weniger anstrengend sondern eher wertvoll für alle sein.

#### **7. Stetige Erweiterung des Softwareangebots**

---

<sup>13</sup>quelle Waymo

Die Bereitstellung eines Softwareshops ist damit gekoppelt, dass eigene und externe Entwicklerteams stetig neue Softwares veröffentlichen und bestehende weiterentwickeln. Die Weltweit ca. 1,3 Milliarden registrierten Kraftfahrzeuge [15, Vgl.] bieten eine sehr große Kundenbasis und somit auch eine gute Einnahmequelle, welche viele Entwicklerteams anlocken können. Das Spektrum neuer Softwares kann durch die steigende Anzahl an Entwicklern stark zunehmen und Anwendungsfälle können schneller abgedeckt werden.

## 8. Interaktion mit der Autoumwelt

Bestimmte Softwares können die Kommunikation mit anderen Akteuren der Fahrzeugumwelt (*Am-peln, Parkplätze, andere Fahrzeuge, Drive-In-Restaurants, uvm.*) ermöglichen und so neue Möglichkeiten zur Interaktion untereinander schaffen und auch die Effizienz des Verkehrs nachhaltig durch V2X-Kommunikation steigern. [25, Vgl. S.19]

### 3.1.5 Einnahmequellen

Ein Softwareshop hat zwei unterschiedliche Einkommensströme. Zum einen kann es eine 'Anmeldegebühr' geben, die Softwareprovider zahlen müssen um Software im Shop veröffentlichen zu können, wie es auch für den Google Play Store der Fall ist. Außerdem können diese ihre Softwares im Shop bewerben, indem sie Werbeflächen kaufen auf welchen ihre Softwares zum Kauf hervorgehoben werden. Dies ist vergleichbar mit dem Ergebnis einer Google-Suche, bei der ganz oben *gesponserte* Internetseiten/Produkte zu finden sind.

Neben Einnahmen durch die Softwareprovider wird auch Umsatz durch den Verkauf und die Nutzung von Software generiert. Beim Kauf einer Software geht ein fester Prozentsatz des Verkaufspreises an den Shop Betreiber zurück. Zum Vergleich: Bei Android Apps liegt der Anteil den Google einnimmt bei 30%. Auch bei entstehenden In-App-Käufen (*z.B. für die Nutzung eines Services*) geht ein Anteil von 10% an Google.

Durch die Entwicklung von Entwicklungs- und Analyse-Tools können weitere Einnahmen generiert werden. Software Provider können diese kaufen und nutzen, wodurch sie bei der Entwicklung neuer Softwares unterstützt werden.

### 3.1.6 Schlüsselressourcen

Die wichtigsten Ressourcen sind die **geschaffene Plattform des Softwareshops** und die dort vertriebenen **Softwares**. Um eine große Menge an Softwares und sonstigen Daten von Fahrzeugen speichern und analysieren zu können, ist vor allem ein starkes und effizientes Backend wichtig, welches zugleich für die Suche als auch für die Verteilung von Software zuständig ist. Damit der Faktor der Ortsunabhängigkeit realisiert werden kann, ist ein stabiles Kommunikationsnetzwerk (z.B. 5G [25, S. 10]) notwendig.

Da Softwares zum Großteil von Software Providern bereitgestellt werden und der Shop ohne das stetige hinzufügen und aktualisieren von Softwares an Attraktivität verliert ist eine gute Beziehung zu diesen essentiell.

### 3.1.7 Schlüsselaktivitäten

Die Schlüsselaktivitäten sind die Aufgaben des Unternehmens, die zur Erfüllung vorgestellter Nutzenversprechen führen. Sie sind an die Schlüsselressourcen und Nutzenversprechen gekoppelt und sollen die Anforderungen dieser erfüllen. Die Schlüsselaktivitäten stellen im Kontext einer Wert-

schöpfungskette **die wichtigsten Bausteine** dieser dar. Zunächst werden sie daher in vier Gruppen unterteilt, welche die Rolle der jeweiligen Bausteine in der Wertschöpfungskette verdeutlichen sollen. In Kapitel 3.2 werden sie näher erläutert und in einen logischen Kontext gebracht.

### 1. Aktivitäten der Bedarfserkennung von Software

- Software Klassifizierung
- Automatische Erkennung von Softwarebedarf

### 2. Aktivitäten der Bereitstellung von Software

- Sicherheitsverifikation von Software anhand von Sicherheitskonzepten
- Sicherer Download über das Internet
- Anbindung von elektronischen Zahlungsschnittstellen
- Angebotsunterbreitung
- Eigenständige Softwareentwicklung

### 3. Aktivitäten des generellen Betriebs

- Shop Verwaltung
- Shop (*Weiter-)Entwicklung und Wartung*
- Eigenständige Verwaltung von Software durch den Kunden
- Eigenständige Überwachung von Software durch den Kunden
- Kundenservice und individuelle Beratung

### 4. Aktivitäten zur Verbesserung der Supply Chain

- IDE(*Entwicklungsumgebung*) entwickeln
- Dokumentation & API Reference
- Tool Entwicklung (*Für Entwickler*)

#### 3.1.8 Schlüsselpartner

Die Schlüsselpartner sind die Teilnehmer der Supply Chain, von denen der Software Shop direkt abhängig ist. [16, Vgl.] Eine gute Beziehung zu Ihnen ist wichtig, um die aufgestellten Nutzenversprechen erfüllen zu können.

##### • Software Provider

Software Provider stellen die Softwares her, die im Shop zur Verfügung gestellt werden und sind daher die wichtigsten Lieferanten des Shops. Sie sollten bei der Entwicklung von Softwares unterstützt werden, da ohne Softwares kein Shop möglich ist. Die entwickelten Softwares dürfen **keine Sicherheitslücken enthalten** und sollten für möglichst viele Automarken und -modelle verfügbar sein. Die Einbeziehung von ihnen in den Entwicklungsprozess des Shops und von Software APIs kann ihr Verständnis und ihre Fähigkeiten stärken und somit zum Erfolg des Shops beitragen.

- **Automobilhersteller**

Damit die entwickelten Softwares auf Fahrzeugen installiert werden können, müssen Automobilhersteller als Partner gewonnen werden damit diese den Shop in ihre Fahrzeuge integrieren. Je mehr Automobilhersteller (*OEMs*) an dem Shop beteiligt sind desto größer ist dessen Reichweite, wodurch auch die Attraktivität dessen für neue Software Provider steigt. Die Einbeziehung möglichst vieler OEMs soll verhindern, dass ein Konkurrenzprodukt des Software Shops entsteht.

Integriert ein OEM den Shop in seine Fahrzeuge, muss dieser die Entwicklung von APIs für die eigenen Fahrzeugmodelle vorantreiben, damit Softwares auf diesen ausgeführt werden können.

- **Mobilfunkbetreiber**

Damit Softwares orts- und zeitunabhängig heruntergeladen werden können, müssen Autos eine kontinuierliche Verbindung zum Internet haben. Wenn Fahrzeuge jedoch nicht in der eigenen Garage stehen ist es eher unwahrscheinlich, dass sie in mit einem WLAN verbunden sind. Durch ein stabiles und gut ausgebautes mobiles Datennetz kann ein Fahrzeug auch ohne die Verbindung mit einem WLAN oder anderen Netzwerkstrukturen Softwares herunterladen. Damit die Kosten für Fahrzeughalter hierbei skalierbar gehalten werden, ist eine Partnerschaft mit Mobilfunkanbietern sinnvoll mittels welcher der Internetzugriff für Softwaredownloads günstiger Werden kann.

- **Elektronische Bezahlsysteme**

Um den Kauf von Softwares einfach zu gestalten, ist die Einbindung elektronischer Bezahlsysteme wie Paypal, MasterCard, Klarna oder sonstigen wichtig. Hierdurch kann der Kauf von Softwares schnell als auch orts- und zeitunabhängig abgeschlossen werden.

- **Service Provider**

Von Service Providern ist der Shop zwar nicht abhängig, aber durch sie können viele Mehrwerte für Fahrzeughalter geschaffen werden die sonst nicht existieren würden. Sie bieten Services an, die von Fahrzeughaltern mit Hilfe einer Software in Anspruch genommen und genutzt werden können.

### **3.1.9 Kostenstruktur**

Die Pflege der Beziehungen mit den Schlüsselpartnern ist wichtig, da der Software Shop ohne sie nicht existieren könnte. So kann die Entwicklung einer IDE, von Tutorials, API Referenzen und Analystools für Software Provider zugleich wichtig als auch kostspielig sein. Auch sonstige Events des Unternehmens wie Konferenzen und Tagungen, welche zur Bindung von Schlüsselpartnern dienen, sind mit hohen Kosten verbunden.

Damit die Integration des Shops in die Fahrzeuge von Automobilherstellern erfolgt ist es wahrscheinlich, dass Automobilhersteller prozentuale Gewinne des Shops erhalten. Diese Summe könnte anhand der Menge verkaufter Softwares auf Fahrzeugen dieser Marke berechnet werden. Weiterhin fallen Kosten für den Betrieb, die (Weiter-)Entwicklung und die Vermarktung des Shops an. Diese sind im wesentlichen für die Bezahlung von Managern, Entwicklern und Designern. Weiter fallen Stromkosten für das Betreiben von Servern an.

Das Business Model Canvas hat einen Überblick über die wesentliche Aspekte einer Unternehmung geboten, welche neue Fahrfunktionen für intelligente Fahrzeuge bereitstellen möchte. Es bietet Ansätze für Umsetzung in der realen Welt und liefert wichtige Erkenntnisse für die im folgenden aufgestellte Wertschöpfungskette.

### 3.2 Die Wertschöpfungskette

Damit der Weg von Entwicklung, über Bereitstellung und letztendlichen Kauf einer Software nachvollzogen werden kann, werden einige im Business Model identifizierten Schlüsselpartner in Abbildung 7 im Kontext einer Supply Chain dargestellt. Der Prozess startet bei Software Providern, welche eine Idee für eine Software haben und diese umsetzen. Hierzu ist möglicherweise die Zusammenarbeit mit Service Providern notwendig. Die entwickelte Software wird an den Software Shop weitergeleitet und dieser stellt sie entweder im Shop bereit oder nicht. Abschließend können Fahrzeughalter Software über eine kabellose Schnittstelle wie das Internet herunterladen, wobei Mobilfunkbetreiber der Mittelsmann für Mobilfunknetze darstellen. Die Wertschöpfungskette ist als detailliertere Sicht des "Software Shops" (Abbildung 7) zu sehen. Sie stellt die Entscheidungen und Schritte dar, die eine Software von der Lieferung des Software Providers (**IN**) bis zum Download im Shop **OUT** durchlaufen und bestehen muss.

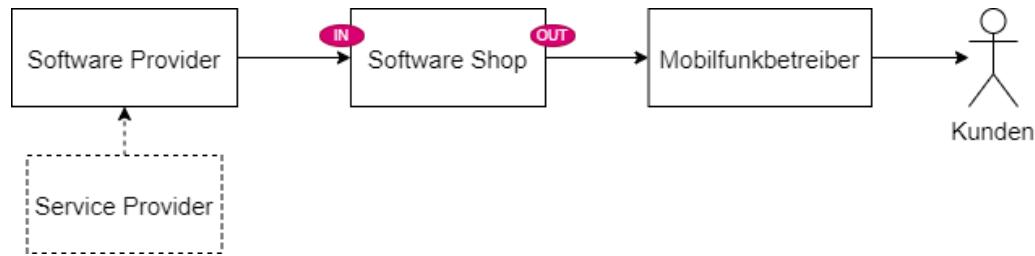


Abbildung 7: Supply Chain von Software

Nach Porter ist die Wertschöpfungskette in primäre und unterstützende Aktivitäten aufzuteilen. [21] Primäre Aktivitäten "liefern [dabei] einen direkten wertschöpfenden Beitrag zur Erstellung eines Produktes" [21, ] und Unterstützende Aktivitäten sind als "notwendige Voraussetzung zu Erstellung der Produkte" [21, ] zu sehen. Sie sind im Rahmen aller Primäraktivitäten aktiv und beeinflussen die Qualität des Produktes [18, Vgl.]. Abbildung 8 ordnet die in Kapitel 3.1.7 identifizierten Bausteine den Aktivitäten einer Wertschöpfungskette zu - die dort festgelegte Gruppierung der Schlüsselaktivitäten wird beibehalten.

#### 3.2.1 Primäre Aktivitäten

Es gibt fünf primäre Aktivitäten. Die erste Aktivität ist die Beschaffung und Lagerung von Materialien, in diesem Fall von Softwares, welche als **Eingangslogistik** bezeichnet wird (*Links unten im Bild*). Im Kontext dieser sind zwei wichtige Bausteine zu erwähnen:

## Verifikation der Software-Sicherheit

Sämtliche Softwares die dem Shop hinzugefügt werden sollen, müssen zunächst diverse Sicherheitschecks bestehen. Zum einen sollten Softwares hinsichtlich aufgenommener und gespeicherter Daten überprüft werden. Je nach Land gelten unterschiedliche Datenschutzrichtlinien, welche zu beachten sind um die Privatsphäre von Fahrzeughaltern zu schützen. Weiterhin muss festgestellt werden, dass durch die neue Software keine Sicherheitslücken bei der Fahraufgabe entstehen. Dies kann durch Simulationen getan werden, wie in Kapitel 4 gezeigt wird.

Es ist sinnvoll, ein Sicherheitskonzept zu entwickeln in welchem die zum einen die aufgezählten als auch weitere mögliche Sicherheitslücken von Softwares geprüft werden. Das Ziel der Sicherheitsverifikation ist letzten Endes, dass durch neues Softwares keine Gefahren für die Fahrzeuginsassen entstehen.

## Klassifizierung von Software

Erfüllt eine Software die Sicherheitsrichtlinien, wird sie anschließend Klassifiziert. Die Klassifizierung von Software soll die Suche und Darstellung dieser im Shop unterstützen, in dem einer Software diverse Meta-Daten hinzugefügt werden. Diese können im Shop einerseits als Such-Filter verwendet werden, einzelne können auch angeben wie gut/schlecht eine Software ist. Ein beispielhaftes Konzept wird in Kapitel 4.1 erarbeitet.

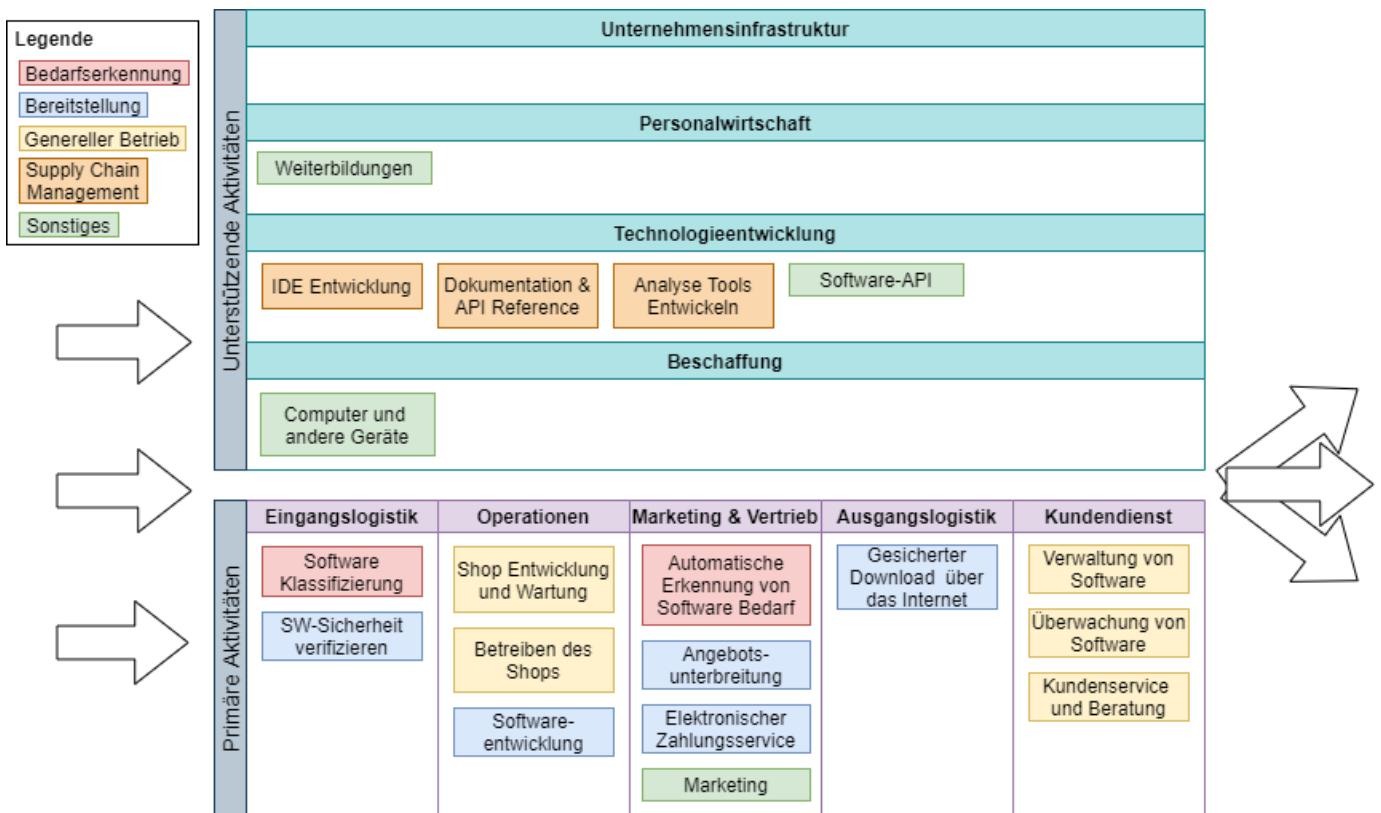


Abbildung 8: Die wichtigsten Bausteine einzelner Aktivitäten der Wertschöpfungskette

Nach erfolgreichen Durchlaufen der Eingangslogistik folgen die Aktivitäten der **Operation**. Im Rahmen dieser werden die verifizierten und klassifizierten Softwares dem Shop hinzugefügt und verwaltet. Die Operationen sind die Aufgaben des Unternehmens, in welchen der eigentliche Mehrwert für die Kunden geschaffen wird.

## **Shop (Weiter-)Entwicklung und Wartung**

Der Software Shop ist das Herz des Unternehmens, ohne welchen dessen Existenz nicht möglich ist. Die (Weiter-)Entwicklung und Wartung des Shops ist daher eine der Kernaufgaben des Unternehmens. Änderungen und neue Funktionen betreffen entweder das Front-End oder das Back-End des Shop. Das Front-End ist die grafische Oberfläche (*Mensch Maschine Schnittstelle*), die letzten Endes von den Fahrzeughaltern bedient wird. Bei der Front-End-Entwicklung kann die Verwendung des User-Centered-Designs zu einer Verbesserung von UI und UX (*User-Experience*) führen. Das Back-End ist für den Fahrzeughalter nicht sichtbar - es **verarbeitet** **Kauf- und Suchanfragen für Softwares** und muss dementsprechend schnell und effizient sein, um eine große Menge derartiger Anfragen zu bewältigen. Da der Shop viel Software umfasst, die von Software Providern entwickelt wurde, sollte bei der (Weiter-)Entwicklung des Shops auf Kritik, Verbesserungsvorschläge und Wünsche dieser eingegangen werden. Auch die übrige *Community* (OEMs, Fahrzeughalter, Flottenbetreiber) sollte hier einbezogen werden. Neben der (Weiter-)Entwicklung des Shops, muss auch die bereits existierende Plattform zusätzlich gewartet werden. Dies umfasst den Austausch von unzuverlässiger Hardware, aber auch das schnelle Beheben von Bugs, die im Shop aufgetreten sind sowie die kontinuierliche Überwachung der Datenbanken, Server etc.

## **Betreiben des Shops**

Damit der Shop betrieben werden kann, müssen durch die Eingangslogistik klassifizierte und verifizierte Softwares dem Shop hinzugefügt oder aus diesem entfernt und eingehende Download-Anfragen verarbeitet werden. Außerdem müssen Softwares die vermehrt von Fahrzeughaltern gemeldet wurden, müssen überprüft und gegebenenfalls aus dem Shop entfernt werden. Damit im Shop keine Inhalte (*Softwares, Bewertungen von Softwares*) vorkommen, die diskriminierend, rassistische oder auf eine andere Art & Weise Menschenrechts-verletzend sind, sollten diese sofort aus dem Shop entfernt werden

. Softwares die im Laufe der Zeit als „besonders gut“ aufgefallen sind, können ein Siegel „Empfehlung der Redaktion“ erhalten. Dieses soll die Qualität einer Software hervorheben und Fahrzeughalter bei der Kaufentscheidung unterstützen.

## **Eigene Softwareentwicklung**

Neben der Bereitstellung von Software anderer Software Provider ist auch die eigene Entwicklung und Bereitstellung von Software im Shop möglich. Einerseits stellt der Verkauf eigener Software weitere Einnahmequelle für das Unternehmen, andererseits lässt sich durch den Verkauf einer guten, günstigen Software das Image des Shops steigern.

Die vorgestellten Aufgaben stellen Voraussetzungen dar, welche für den Verkauf von Software notwendig sind. Da der Shop bereits auf Fahrzeugen installiert ist (*siehe Abschnitt ??*), müssen Fahrzeughalter anschließend zum Kauf bewegt werden. Dies passiert durch den dritten Aufgabenbereich der Wertschöpfungskette, dem **Marketing und Vertrieb** von Software.

## **Automatische Erkennung von Softwarebedarf**

Um Fahrzeughalter beim Kauf von Software zu unterstützen soll sichergestellt werden, dass gekaufte Softwares auch tatsächlich benötigt werden. Hierzu soll der Softwarebedarf eines Fahrzeug automatisch anhand von Fahrt- und Fahrzeugdaten identifiziert und dem Fahrzeughalter vorgeschlagen werden. Es gibt diverse Möglichkeiten den Softwarebedarf eines Fahrzeugs automatisch zu bestimmen:

### **Suche anhand einer einzelnen Situation**

Diese Suche wird in Kapitel 2.2.3 beschrieben. Das Ergebnis der Suche enthält entweder **keine** oder genau **eine** Software, welche eine zuvor identifizierte Lücke der autonomen Fahr-funktionen abdeckt. Die Suche soll automatisch erfolgen und die Vorschläge werden im Shop angezeigt.

### **Routen-Basierte Suche nach Software**

Ebenfalls in Kapitel 2.2.3 beschrieben, erfolgt diese Suche vor Fahrtantritt und durchgeführt wenn das Fahrzeug bzw. der Fahrzeughalter kurz davor ist eine nicht häufig zurückgelegte Strecke zu Fahren. Die Ergebnisse der Suche werden in die Routenplanung integriert und dem Fahrzeughalter wie in Kapitel 2.2.3 beschrieben vor Fahrtantritt vorgeschlagen.

### **Shop-Basierte Suche (*Shop-Auswahl*)**

Wird eine Software im Shop ausgewählt die dem Fahrzeughalter noch nicht vorgeschlagen wurde, kann dieser bestimmen lassen ob und wenn ja zu welchem Umfang die jeweilige Software das Fahrspktrum des Fahrzeugs erweitert. Das Ergebnis der Anfrage stellt eine gute Entscheidungsgrundlage für den Kauf der Software dar, da dem Fahrzeughalter so der tatsächliche *persönliche* Nutzen einer Software präsentiert wird.

### **Umgebungsbedingte Softwaresuche**

Die Umgebungssuche ist eine Weiterentwicklung der in Kapitel 2.2.3 dargestellten *Suche aus Basis der Fahrtenhistorie*. Die Idee dieser ist es, dass dem Fahrzeughalter Softwares vorgeschlagen werden die andere Fahrzeughalter in der Region vermehrt installiert haben. Die Umgebungs-Suche wird in Kapitel 4.2 im Detail erläutert.

Abhängig davon, ob passende Softwares gefunden wurden oder nicht, werden sie dem Fahrzeughalter vorgeschlagen. Durch den Prozess wird die Kundenzufriedenheit gesteigert, als auch mehrere Nutzenversprechen gefördert (*NV-2, NV-3, NV-4, NV-5*). Neben Suchalgorithmen die dem Fahrzeughalter Softwarevorschläge unterbreiten, können auch Service Provider einem Fahrzeug eine Software vorschlagen. Dies ist notwendig, wenn der Fahrzeughalter einen Service nutzen will für welchen eine bestimmte Software auf dem Fahrzeug installiert sein muss.

### **Angebotsunterbreitung**

Wird anhand einer der zuvor aufgelisteten Suchen festgestellt, dass eine bestimmte Software die Fahrfunktionen eines Fahrzeugs sinnvoll erweitern kann, sollte diese dem Fahrzeughalter vorgeschlagen werden. Im Kontext der Angebotsunterbreitung ist der Zeitpunkt wichtig, zu welchem die Software dem Fahrer vorgeschlagen wird. Ein Konzept für die Bestimmung eines geeigneten Zeitpunkts zum Vorschlagen eines Software wurde in Kapitel 2.3.3 erläutert.

### **Bereitstellung elektronischer Zahlungsschnittstellen**

Um den Kauf einer Software abschließen zu können, müssen Fahrzeughalter den Kaufpreis über eine elektronische Zahlungsschnittstelle bezahlen können. Hierzu sollten bestenfalls mehrere gängige Zahlungsschnittstellen (*MasterCard, PayPal, EC-Cash o.A.*) mit dem Shop verknüpft werden, um so mehr Fahrzeughaltern den Kauf zu ermöglichen. Um weitere eigene Umsätze zu generieren, kann hier auch eine eigene Zahlungsschnittstelle eingebaut werden. Fahrzeughalter können zur Verwendung dieser gebracht werden, indem der Kauf mittels dieser Schnittstelle beispielsweise günstiger oder schneller ist.

## **Marketing**

Wie im Business Model beschrieben, ist der Umfang des Marketings maßgebend für den Erfolg des Shops. Die Kampagnen sollten auf möglichst vielen Kanälen zu sehen sein und dabei möglichst viele Alters- und Kundengruppen ansprechen. Vor allem die Vermarktung über Social Media Kanäle ist ratsam. Es kann Sinnvoll sein "Markenbotschafter\*in finden, die Content für Social Media Plattformen entwerfen. Coca Cola hat dies beispielsweise erfolgreich mit Coke TV<sup>14</sup> getan.

Damit die von Fahrzeughaltern gekauften Softwares letztendlich auch an Fahrzeuge verteilt werden können, muss die **Ausgangslogistik** strukturiert werden welche festlegt, über welche Distributio-naskanäle Software verteilt wird.

## **Gesicherter Download über das Internet**

Die Bereitstellung bezahlter Softwares wird mit dem Download abgeschlossen. Dieser Down-load soll orts- und zeitunabhängig sein und muss über eine sichere Schnittstelle erfolgen. Um dies zu gewährleisten, sollte der Ausbau von Kommunikationsnetzwerken (*bspw. 5G*) geför-dert werden und eine sichere Architektur für den Server entworfen werden (*Uptane, Kapitel 2.3.1*). Da durch den Download von Software über das Internet hohe Kosten für Fahrzeughal-ter entstehen können, sollten Kooperationen mit Mobilfunkanbietern abgeschlossen werden, durch welche die Kosten skalierbar gehalten und die Kundenzufriedenheit gesichert werden kann.

Um die Kundenzufriedenheit weiter zu stärken, müssen auch nach dem Kauf für den Kunden wertschöpfende Aktivitäten durchgeführt werden. Diese werden dem letzten Schritt der Wertschö-pfungskette, dem **Kundendienst**, zugeordnet.

## **Verwaltung von gekaufter Software**

Damit Fahrzeughalter wissen, welche Softwares sie gekauft, geliehen oder gemietet haben soll für sie die eigenständige Verwaltung von Software im Shop möglich sein. Fahrzeughalter behalten hierdurch den Überblick über ihr Fahrzeug und können Softwares entfernen oder hinzufügen.

## **Überwachung installierter Software**

Um Fahrzeughaltern vor Augen zu führen, welche Softwares tatsächlich vom Fahrzeug genutzt werden und welche nicht, sollten diese Einblick in eine Art Überwachung der Software haben. Hier können grundlegende Statistiken geführt werden wie

- die durchschnittliche Nutzungsdauer einer Software je Woche
- das Datum, an welchem die Software das letzte mal genutzt wurde
- oder eine Liste, in welcher zu sehen ist welche Softwares an welchem Tag der Woche vermeintlich genutzt werden.

Aufgrund der steigenden Bedeutung des Datenschutzes könnte auch eine Ansicht erstellt wer-den, welche zeigt welche Daten von Softwares erhoben werden. Hierdurch könnte das Vertrau-en von Kunden gestärkt werden (*Oder auch nicht, je nach dem welche und wie viele Daten erhoben und gespeichert werden*).

---

<sup>14</sup>[https://www.youtube.com/channel/UCTfwo-EWSD-8hZ7F8HG\\_qJg](https://www.youtube.com/channel/UCTfwo-EWSD-8hZ7F8HG_qJg), Aufgerufen am 14. Juni 2020

## Kundenservice und Beratung

Um vor allem die Bindung zu älteren Kunden zu stärken, kann das einrichten einer Telefonischen Beratung sinnvoll sein. Vor allem bei der Ersteinrichtung eines Fahrzeugs kann die telefonische Unterstützung von Fahrzeughaltern positive Auswirkungen auf die Kundenzufriedenheit haben. Auch alternative, vorwiegend digitale Varianten Fahrzeughalter bezüglich Softwares zu beraten kann einen Mehrwert für diese bieten. Durch das entwickeln von Nutzerforen oder der Möglichkeit, Softwares im Shop zu bewerten (*1-5 Sterne + Kommentar*) können Fahrzeughalter sich gegenseitig beraten.

## Überblick

Die fünf Aktivitäten der Wertschöpfungskette schaffen Mehrwerte für Fahrzeughalter. Wird eine Software von einem Software Providern beim Softwareshop „eingereicht“, wird sie für den Verkauf an Fahrzeughalter vorbereitet. Jeder einzelne Schritt der Wertschöpfungskette trägt dazu bei, dass Fahrzeughalter letztendlich selbstständig Softwares kaufen können, die sie tatsächlich benötigen. Während all diese Schritte durchlaufen werden, fügen die *unterstützenden Aktivitäten* stetig Mehrwerte zur Wertschöpfung hinzu und sind daher *indirekt am Erfolg beteiligt*. Im folgenden werden diese vorgestellt und ihre Rolle im Ablauf der Wertschöpfungskette verdeutlicht.

### 3.2.2 Unterstützende Aktivitäten

Es gibt vier Unterstützende Aktivitäten. Angefangen bei der Schaffung einer gut aufgebauten **Unternehmensinfrastruktur**, welche alltägliche Problematiken beheben und Zeit sparen kann. Zu dieser gehört ein gute Anbindung zur Außenwelt, schnelles Internet und weiteres.<sup>15</sup> Neben der Schaffung einer guten Unternehmensinfrastruktur hat auch die **Personalwirtschaft** (*Human-Resource-Management*) einen starken Einfluss auf den Erfolg des Unternehmens. Sie beschreibt wichtige, die Mitarbeiter unterstützende Strukturen und Regelungen des Unternehmens, wie zum Beispiel Weiterbildungen und Workshops.

#### Weiterbildungen

Die kontinuierliche Weiterbildung der Mitarbeiter ist für den Erfolg des Shops überaus wichtig. Weiterbildungen müssen nicht Zwangswise in dem jeweiligen Fachbereich des Mitarbeiters liegen, sondern können auch neue Aspekte beleuchten. So können Entwickler einerseits gemeinsamen neue Programmiersprachen & -frameworks lernen, anderseits können auch andere „Skills“ beigebracht oder Aktivitäten (*bspw. Sport*) miteinander durchgeführt werden, welche die Team-Chemie steigern. Die Schaffung einer guten Work-Life-Balance kann die Produktivität der Mitarbeiter erhöhen [23, Vgl. S.4] und somit den Erfolg des Shops nachhaltig unterstützen.

Für ein (digitales) Unternehmen ist die stetige **Technologieentwicklung** wichtig und kann wegweisend für den Erfolg des Unternehmens sein.?? Zu dieser gehört nicht nur die Schaffung von Werten für die innerbetriebliche Wertschöpfungskette, sondern auch die Supply Chain übergreifend Unterstützung der Partner und Kunden.

#### IDE Entwicklung

Damit Software Provider bei der Entwicklung von Softwares unterstützt werden, ist die Bereitstellung einer eigens entwickelten Entwicklungsumgebung(*IDE*) sinnvoll. Diese IDE kann Features und Tools umfassen, welche explizit für die Entwicklung von (Fahrzeug-) Softwares

---

<sup>15</sup>quelle

entwickelt wurden. Durch integrierte Simulatoren, Sicherheitschecks und andere Tools könnte der Weg von Idee über Entwicklung hin zu Bereitstellung im Shop verkürzt und vereinfacht werden. Ein Vergleich aus der Realität ist Android Studio<sup>16</sup> aber auch die Eclipse IDE, welche in Zusammenarbeit mit Unternehmen aus der ganzen Welt entwickelt wird<sup>17</sup> und die Einbindung von Plugins einfach gestaltet.

### Dokumentation und API-Reference

Die Entwicklung von Software wird zusätzlich durch die Bereitstellung einer ausführlichen API-Reference sowie technischen Dokumentation (mit Beispielen) positiv unterstützt. Eine API-Reference sollte alle Funktionen des Frameworks ausführlich beschreiben und im Nutzungskontext vorstellen. Diagramme unterstützen zusätzlich das Verständnis der API. Um die Entwicklung neuer Softwares voranzutreiben und zu vereinfachen ist es sinnvoll eine Einführung (*'Getting Started Guide'*) für diese bereitzustellen. Google stellt in diesem Kontext das *Android Jetpack*<sup>18</sup> sowie weitere Dokumentationen mit Anleitungen und Videos zur Verfügung.<sup>19</sup> Auch und vor allem sollten hier die Sicherheitskriterien, welche im Kontext der Wertschöpfungskette in Abschnitt **Eingangslogistik** erwähnt wurden, vorgestellt werden.

### Entwicklung von Analyse-Tools

Um Software Provider zusätzlich bei der (Weiter-)Entwicklung zu unterstützen, können Analyse-Tools entwickelt werden, welche die Entwickler Entwicklung unterstützen. Analysierte Statistiken können generelle Kennzahlen (*Nutzung in H je Tag*) aber auch spezifischere Werte (*Button-Clicks*) sein, durch welche Entwickler dazu in der Lage sind, ihre Software zu verbessern.

### API-Entwicklung

Damit Software Provider bestimmte Systeme von Fahrzeugen ansteuern können, müssen APIs für die jeweiligen Systeme eines Fahrzeugs entwickelt werden. Über diese können Daten der Systeme ausgelesen oder eingefügt werden. Da künftig hergestellte Fahrzeuge sehr wahrscheinlich neuartige Technologien und Feature enthalten die heutzutage noch nicht existieren, müssen die APIs im Laufe der Zeit immer weiter angepasst bzw. erweitert werden.

## ANPASSEN und API-KONZEPT EINBAUEN!

Damit „alte“ Fahrzeuge weiterhin von einer Software gesteuert werden können, müssen die alten APIs aber bestehen bleiben. In Android wird diese Problematik sinnvoll durch das *API-Level* gelöst: Jede Android Version hat ein eigenes *API-Level*. Je höher das Level, desto höher auch die Android Version. Ein Smartphone mit API-Level 23 (*Android 6.0*) kann alle Funktionen und Methoden *„tieferer“* APIs nutzen, nicht aber die *„höherer“* APIs ( $> 23$ ). Jede App hat eine *minimale API-Version* welche festlegt, welches API-Level das Endgerät mindestens haben muss. Im Kontext von Fahrzeugen ist es vorstellbar, dass jedes Teilsystem eines Fahrzeugs ein eigenes minimales API-Level hat. Softwares dadurch nur auf Fahrzeugen installiert werden, welche den Anforderungen der Software entsprechen.

Die letzte unterstützende Aktivität ist die **Beschaffung**. Ihre Aufgabe ist, dass alle Mitarbeiter mit den nötigen Materialien wie Computern, Tastaturen, Stiften oder anderem versorgt werden.

---

<sup>16</sup><https://developer.android.com/studio>

<sup>17</sup><https://www.eclipse.org/org/>, Aufgerufen am 21. Juni 2020

<sup>18</sup><https://developer.android.com/jetpack>, Aufgerufen am 21. Juni 2020

<sup>19</sup><https://developer.android.com/guide>, Aufgerufen am 21. Juni 2020

### **3.3 Zusammenfassung**

Das in Kapitel 3.1 erstellte Business Model Canvas bietet einen guten Überblick der Aufgaben eines Software Shops und der damit einhergehenden Kosten, Partner und aufzubauenden Strukturen. Es stellt dar, was Unternehmen bei der Erstellung eines Software-Shops beachten müssen wie gewisse Mehrwerte für Fahrzeughalter geschaffen werden können. Die Auflistung von Nutzenverprechen, Schlüsselpartnern und sonstigen Grundlagen des Shops wäre durch eine Wertschöpfungskette alleine nicht möglich gewesen. Der geschaffene Überblick des Shops hat die anschließende Erstellung der Wertschöpfungskette erleichtert, da einzelne Aspekte der Bedarfserkennung und Bereitstellung von Software ohne weitere Erklärungen eingebaut werden konnten.

Durch die Einordnung der identifizierten Schlüsselaktivitäten bzw. Bausteine in eine Wertschöpfungskette wurde veranschaulicht, wie diese miteinander zu verknüpfen sind um eine Wertschöpfung für den Kunden zu erreichen. Es wurden Abhängigkeiten und Rollen einzelner deutlich, wie zum Beispiel die Rolle der Sicherheitsverifikation als „*Gatekeeper*“ des Shops. Durch die Integration von Bausteinen des *generellen Betriebs*, des *Supply Chain Managements* und sonstigen, welche alle nur indirekt mit der Bedarfserkennung und Bereitstellung von Software zu tun haben, ist die Wertschöpfungskette vollständiger und realistischer geworden.

Um im Prototypen den Weg von Bedarfserkennung über Bereitstellung hin zur eigentlichen Nutzung von Software abbilden zu können werden im weiteren Verlauf technische Konzepte vorgestellt.

## 4 Technische Konzepte

Im kommenden Kapitel wird zunächst ein mögliches Konzept für die *Klassifizierung von Software* vorgestellt, welche in der Eingangslogistik stattfindet. In Kapitel 4.2 wird ein Such-Algorithmus skizziert, welcher die *automatische Erkennung von Softwarebedarf* unterstützt. Im letzten Kapitel werden drei Kommunikationskonzepte vorgestellt, welche den Ablauf der Kommunikation zwischen dem Fahrzeug, dem Server und Service Providern im Kontext automatischen Parkens skizzieren und im Prototypen verwendet werden.

### 4.1 Klassifizierung von Software

Das Ziel der Software Klassifikation ist es, den Fahrzeughaltern beim Kauf unterstützende Kennzahlen zur Verfügung zu stellen, anhand welcher sie sinnvollere Kaufentscheidungen treffen können und somit potentiell einen größeren Wert von der gekauften Software erhalten.

#### 4.1.1 Berechtigungen

Eine dieser Kennzahlen ist die Anzahl genutzter Berechtigungen des Fahrzeugs. Nicht nur die Anzahl, sondern auch der Umfang einzelner Berechtigungen können Kunden vom Kauf abschrecken. [27, Vgl. S. 107] Eine Berechtigung erteilt einer Software die Erlaubnis, auf einzelne Systeme bzw. APIs des Fahrzeugs zugreifen zu dürfen und steigern hierdurch die Sicherheit dessen. Sie werden von Entwicklern zu einer Software hinzugefügt und müssen beim Kauf die Zustimmung des Fahrzeughalters erhalten, damit diese im vollen Umfang vom Fahrzeug genutzt werden kann. Stimmen Fahrzeughalter Berechtigungen nicht zu, kann die entsprechende Software nicht gekauft werden. Durch die Verwendung von Berechtigungen kann die Sicherheit von Softwares gesteigert werden da sie verhindern, dass eine Software ohne weiteres auf alle Daten und Systeme des Fahrzeugs zugreifen kann. Im Kontext autonomer Fahrzeuge können mögliche Berechtigungen von der Architektur dieser abgeleitet werden, welche in Abbildung 9 dargestellt wird.

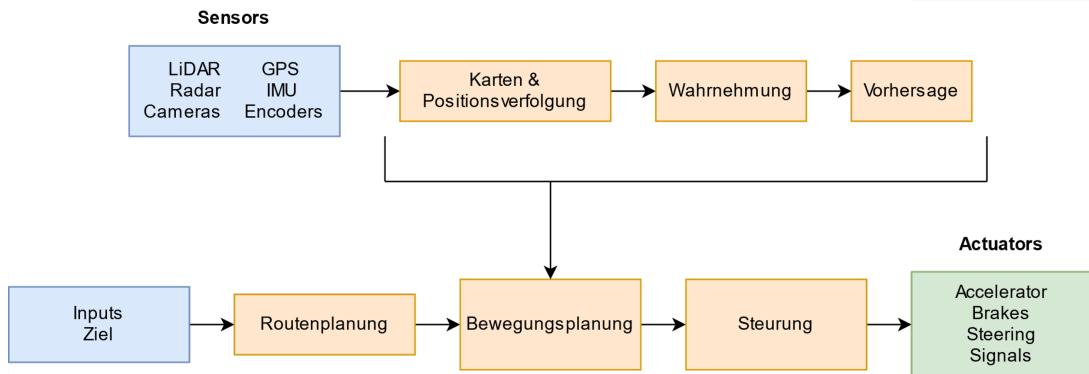


Abbildung 9: Jeff Schneider: Architecture of Autonomous Vehicles [5]

Damit Berechtigungen die Fahrzeughalter davor schützen persönliche Informationen unfreiwillig zu teilen und einer Software so zu viel "Macht" über sich oder das Fahrzeug zu geben, sollte der Umfang von Berechtigungen möglichst gering gehalten werden. So sollte jede Sensorgruppe des Fahrzeugs eigene Berechtigungen haben, da die Art der aufgenommenen Informationen sich voneinander unterscheidet. Durch das Auslesen des GPS-Sensors kann die Position des Fahrzeugs kontinuierlich aufgezeichnet werden, während dies über den Radar nicht möglich ist. Würde alle

Sensoren über die gleiche Berechtigung zugänglich sein, würde dies dementsprechend eine Sicherheitslücke darstellen.

Neben der Privatsphäre der Fahrzeughalter ist vor allem die Sicherheit von Fahrzeuginsassen wichtig. Nimmt eine Software Einfluss auf Entscheidungen, Berechnungen oder sonstigem kann hierdurch eine Sicherheitslücke entstehen. Softwares, die diese Systeme nur auslesen aber nicht direkt beeinflussen haben keine Auswirkung auf die Fahrsicherheit. Daher sollte jedes in Abbildung 9 dargestelltes System jeweils zwei einzelne Berechtigungen haben. Durch die eine kann ein System ausgelesen und durch die andere können Systeme beeinflusst werden.

Neben Berechtigungen, die einzelne Systeme eines Fahrzeugs benötigen, sollten auch weitere Services des Fahrzeugs nur durch die Nutzung von Berechtigungen möglich sein. Eine Berechtigung für die Nutzung des Internets ist sinnvoll, da Softwares hierdurch nicht wahllos Nutzerdaten speichern können. Auch die Kommunikation mit anderen Akteuren des Straßenverkehrs sollte über eine Berechtigung beschränkt werden, um den Fahrer vor 'Spam' zu schützen.

#### 4.1.2 Softwarekennzahlen

Neben der Art und Anzahl genutzter Berechtigungen einer Software, können auch andere Kennzahlen einer Software den Fahrzeughalter bei der Kaufentscheidung unterstützen. Einige dieser Kennzahlen können von den im Businessmodel beschriebenen Nutzenversprechen abgeleitet werden (*Kapitel 3.1.4*).

Um darzustellen, dass die Lebenszeit des Autos verlängert wird (*NV-4*) kann ein Wert berechnet werden, der aussagt wie gut eine Software die einzelnen Autoteile (*Motor, Gertiebe, Reifen etc.*) schont. Die Berechnung dieses Werts kann in Simulationsumgebungen wie Carla erfolgen. Zunächst müssen zwei Fahrzeuge simuliert werden, von denen das eine die zu prüfende Software installiert hat und das andere nicht. Beide Fahrzeuge sollen die der Software hinzugefügten Szenarien, welche der Software beigelegt wurden, durchfahren und dabei bestimmte Werte wie die Reifenreibung, die Dämpfungsrate oder andere messen. Durch einen Vergleich der gemessenen Werte kann prozentual angegeben werden, wie stark eine Software bestimmte Teile des Fahrzeugs schont **oder auch eben nicht**. Um Fahrzeughaltern vor Augen zu führen wie viel Zeit durch eine Software gewonnen werden kann (*NV-6*), sollte ein Zeitwert berechnet werden, welcher die durchschnittlich autonom zurückgelegte Zeit einer Software bestimmt. Die Berechnung sollte auf Basis realer Nutzungsdaten durchgeführt werden. Neben der durchschnittlichen selbstständigen Fahrzeit einer Software kann auch die Anzahl an Interaktionen mit der Autoumwelt ein ausschlaggebender Faktor für den Kauf einer Software sein. Weiter Kennzahlen können sein:

- 1. Anzahl an Downloads und Deinstallationen**
- 2. Benötigter Speicherplatz der Software**
- 3. Bewertung der Software durch Fahrzeughalter**

Es ist sinnvoll die Bestimmung einzelner Kennzahlen regionsabhängig zu gestalten, da diese sonst irreführend sein können: Nehmen wir an, eine Software soll das Fahren in der Innenstadt ermöglichen. Wohnt der Fahrzeughalter nicht im Umkreis einer Stadt, würde ihm die Software nichts sonderlich viel bringen. Durch die hohe Anzahl gesparter Stunden ist der Fahrzeughalter überzeugt

und kauft sich diese dennoch. Um derartige Fälle zu verhindern, sollt die Berechnungen von Kennzahlen an die jeweilige Region des Fahrzeugs angepasst sein.

Die vorgestellten Berechtigungen und Softwarekennzahlen sollen im Shop zu jeder Software angezeigt werden, damit Fahrzeughalter sich einen Eindruck einer Software aufbauen können der sie bei der Kaufentscheidung unterstützt. Außerdem können Kennzahlen und Berechtigungen auch als Such-Filter im Shop verwendet werden, was die Suche nach bestimmten Softwares im Shop erleichtern kann.

## 4.2 Umgebungsbedingte Softwaresuche

In Kapitel 3.2 wurden mehrere Möglichkeiten aufgezählt, mit denen der Softwarebedarf eines Fahrzeugs automatisch bestimmt werden kann. Die im folgenden vorgestellte Umgebungssuche soll einem Fahrzeughalter Softwares vorschlagen, welche in der Umgebung des Fahrzeugs des öfteren von anderen Verkehrsteilnehmern gekauft oder genutzt werden.

Die Umgebungssuche beginnt mit dem Verschicken einer Nachricht vom Fahrzeug an den Server. In dieser Nachricht ist das Manifest des Fahrzeugs enthalten, auf welchem unter anderem eine Liste aller installierter Softwares des Fahrzeugs zu finden ist. Außerdem werden Fahrdaten verschickt durch welche ersichtlich wird, in welchen Regionen sich das Fahrzeug zuletzt aufgehalten hat. Der Server sucht innerhalb dieser Regionen nach Softwares, die dort von anderen Fahrzeughaltern gekauft wurden und noch nicht auf dem anfragenden Fahrzeug installiert sind. Ist die Suche abgeschlossen wird geprüft, ob die gefundenen Softwares einen Mehrwert für das Fahrzeug bieten. Um dies zu bestimmen, können die einer Software beigelegten Open Scenario Dateien genutzt werden und in einer Simulation durchfahren werden.

Zunächst wird eine virtuelle Instanz des Fahrzeugs anhand des versendeten Manifests erstellt (*Originalinstanz*). Für jede gefundene Software wird eine weitere Instanz des Fahrzeugs erstellt, auf welcher die jeweilige Software zusätzlich installiert wird (*Neue Instanzen*). Nach dem erstellen der virtuellen Fahrzeuge, durchlaufen die Originalinstanz und die neuen Instanzen die jeweiligen Szenarien der Softwares. Durch einen Vergleich der Simulationen soll festgestellt werden, ob eine Software positive, negative oder überhaupt keine Auswirkungen auf das Fahrverhalten des Fahrzeugs in den Szenarien hat. Positiv ist, wenn das Fahrzeug die Situation nun selbstständig durchfahren kann oder dies Ressourcenschonender tut. Negativ ist es, wenn die Situation durch die neue Software nicht weiter durchfahren wird oder die neue Software zum durchfahren der Situation nicht vom Fahrzeug ausgewählt wird. Dies kann bedeuten, dass auf dem Fahrzeug bereits eine Software installiert ist, welche die Situation bereits abdeckt.

## 4.3 Kommunikationsprotokolle

Damit Software heruntergeladen und anschließend genutzt werden kann, müssen das Fahrzeug, der Software Shop und auch potentiell weitere Akteure des Straßenverkehr (*Autos, Ampeln, Service Provider, oA.*) miteinander kommunizieren. Damit diese Kommunikation sicher und geordnet ist, werden drei Kommunikationsprotokolle erstellt die im Prototypen implementiert werden können. Die in den Protokollen gelb gefüllten Ovale repräsentieren einzelne Module des Prototypen, welche in Kapitel 5 vorgestellt werden.

#### 4.3.1 Selbstständiger Softwarekauf

Das in Abbildung 10 dargestellte Protokoll stellt den Kommunikationsablauf beim Kauf einer spezifischen Software dar. Der Fahrzeughalter kann im Shop nach Softwares suchen, indem der Name einer Software gesucht wird oder Suchfilter verwendet werden. Wurde eine Software ausgewählt (1), wird eine Installationsanfrage vom *Software Applikation Manager* an den Server geschickt. Der Server nimmt die Anfrage an und überprüft, ob die Software auf dem Fahrzeug installiert werden kann. Anschließend bereitet der Server ein **Installationspaket** vor (2). In diesem ist die Software und eine aktualisierte Version des Manifests enthalten. Ist das Installationspaket vorbereitet, wird es an das Fahrzeug verschickt und der Fahrzeughalter wird über die Mensch Maschine Schnittstelle über das eintreffen dieser informiert (3). Abschließend wird die Software auf dem Fahrzeug installiert und der Fahrzeughalter bzw. das Fahrzeug kann sie verwenden.

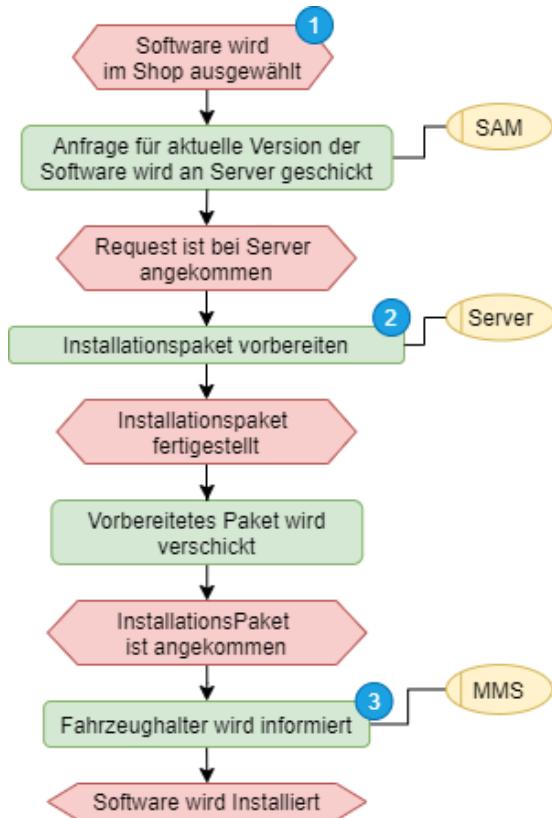


Abbildung 10: Eigenständige Installation von Software

#### 4.3.2 Installationsvorschlag vom Service Provider

In Abbildung 11 wird dargestellt, wie ein Service Provider einem Fahrzeug eine Software vorschlagen kann. Die Notwendigkeit hierfür besteht, da zur Nutzung von Services jeweils eine bestimmte Softwares notwendig sind. Vorzustellen sei folgende Situation: Ein Fahrzeughalter möchte in der Oldenburger Innenstadt einkaufen gehen und muss hierfür sein Fahrzeug parken. Über das Navigationssystem wählt er einen Parkplatz aus den das Fahrzeug daraufhin ansteuert. Um ein Parkticket für diesen Parkplatz kaufen und das Fahrzeug automatisch einparken zu können, wird eine bestimmte Software benötigt. Am Parkplatz angekommen, fährt das Fahrzeug in das Sichtfeld des Parkautomaten, auch *Registrierungszone* genannt (1). Wird ein Fahrzeug erkannt, registriert sich der Parkautomat beim Fahrzeug mittels einer verschickten Nachricht. Ist diese im Fahrzeug angekommen überprüft der Software Applikation Manager, ob die für den Service benötigte Software bereits auf dem Fahrzeug installiert (2).

Ist dies der Fall, wird zusätzlich überprüft ob die installierte Version der Software aktuell ist. Wenn ja, kann der Service genutzt werden - wie die Nutzung eines Service abläuft, wird im untenstehenden Kommunikationsprotokoll 4.3.3 dargestellt. Ist die Software noch nicht installiert oder die installierte Version ist nicht mehr aktuell wird eine Anfrage für die Beschreibung der Software an den Server gesendet, welcher diese folgend verschickt. Der Fahrzeughalter wird im Anschluss über die Mensch Maschine Schnittstelle gefragt ob er die Software kaufen bzw. aktualisieren möchte (4). Er kann das ihm vorgestellte Angebot entsprechend seiner Bedürfnisse für die Software anpassen (*Leihe statt Kauf, etc.*) und seine Entscheidung durch eine Eingabe in der MMS bestätigen. Wird die Anfrage abgelehnt, wird keine Software installiert. Dem Fahrzeug wird eine Abschiedsnachricht geschickt (6), es verlässt die Registrierungszone wieder und wird nicht auf dem Parkplatz parken. Nimmt der Fahrzeughalter die Anfrage hingegen an, sendet das Fahrzeug eine Installationsanfrage an den Server. Der Server nimmt die Anfrage an und überprüft gemäß des Uptane Standards Absatz 5.3.2.1 [22] die Sicherheit des Fahrzeugs. Kann die Sicherheit nicht gewährleistet werden, wird wieder eine Abschiedsnachricht verschickt (6) und das Fahrzeug wird nicht auf dem Parkplatz parken. Kann die Sicherheit gewährleistet werden, erstellt der Server ein Installationspaket für die entsprechende Software (7). Das erstellte Paket wird an das Fahrzeug gesendet, auf diesem installiert und das Fahrzeug kann den angebotenen Park-Service nutzen (8).

#### 4.3.3 Nutzung eines Service

Abbildung 12 zeigt den Kommunikationsablauf zwischen einem Fahrzeug und einem Service Provider Actor (*bspw. ein Parkautomat*). Der dargestellte Ablauf findet statt, wenn ein Fahrzeug einen Service nutzen möchte und die aktuelle Version der hierfür notwendigen Software bereits auf dem Fahrzeug installiert ist (1). Zunächst wird überprüft, ob der Service Provider von dem Software Provider verifiziert ist oder nicht (2). Ist er verifiziert wird über die Mensch Maschine Schnittstelle zur Nutzung vorgeschlagen (3) und der Fahrer kann entscheiden, ob er den Service nutzen möchte oder nicht. Die Entscheidung des Fahrers wird an den Parkautomaten gesendet (4), welcher die Entscheidung evaluiert und dem Fahrzeug dementsprechend entweder einen Parkplatz zuweist oder es verabschiedet. Ist der Service Provider nicht verifiziert, wird das Fahrzeug ebenfalls verabschiedet (5). Die jeweilige Nachricht wird an das Fahrzeug gesendet und hier verarbeitet (6). Das Fahrzeug plant die kommende Bewegung und zeigt den Plan über die MMS an. Entweder fährt es zu dem ihm zugewiesenen Parkplatz oder es verlässt die Registrierungszone wieder.

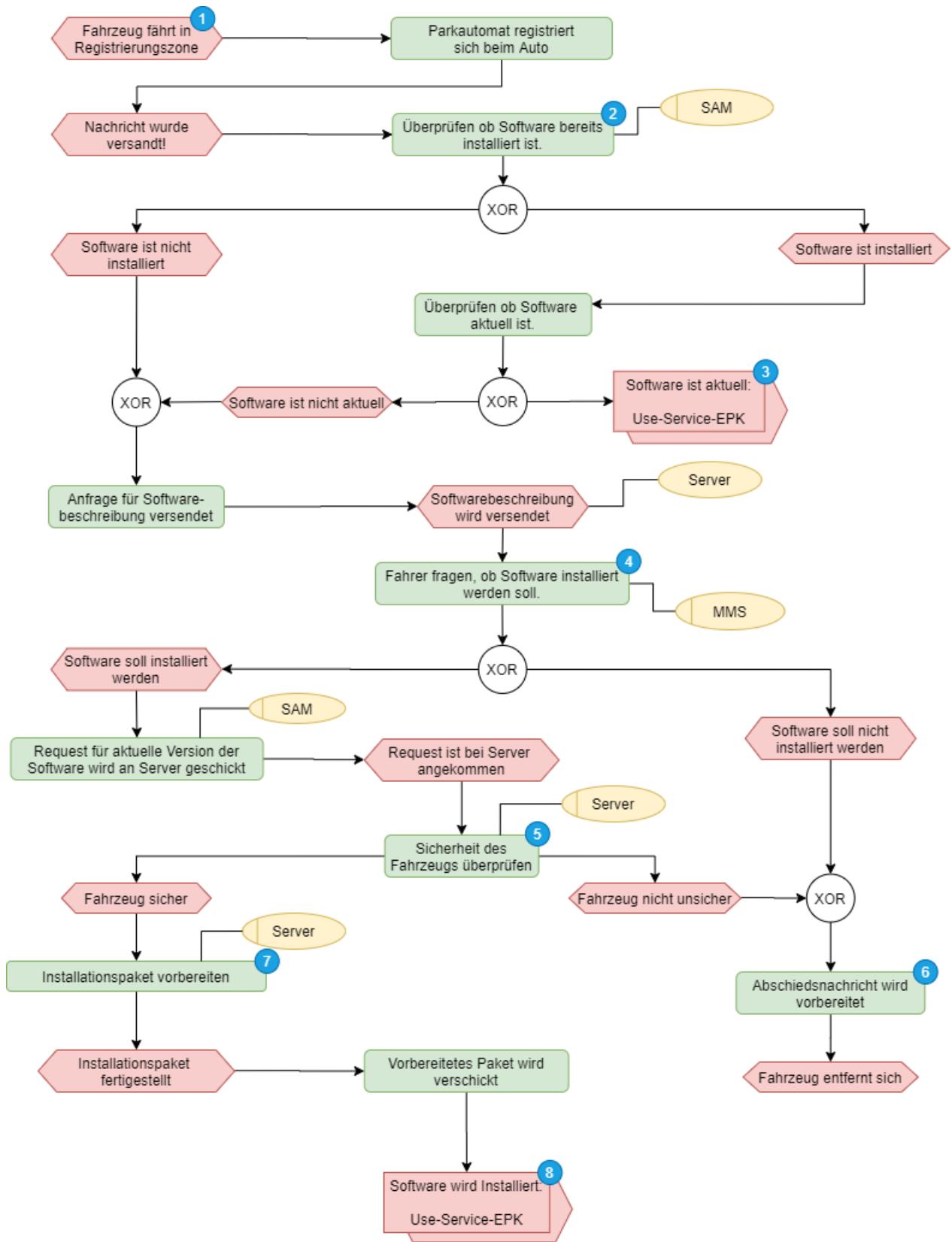


Abbildung 11: Installationsprozess von Software

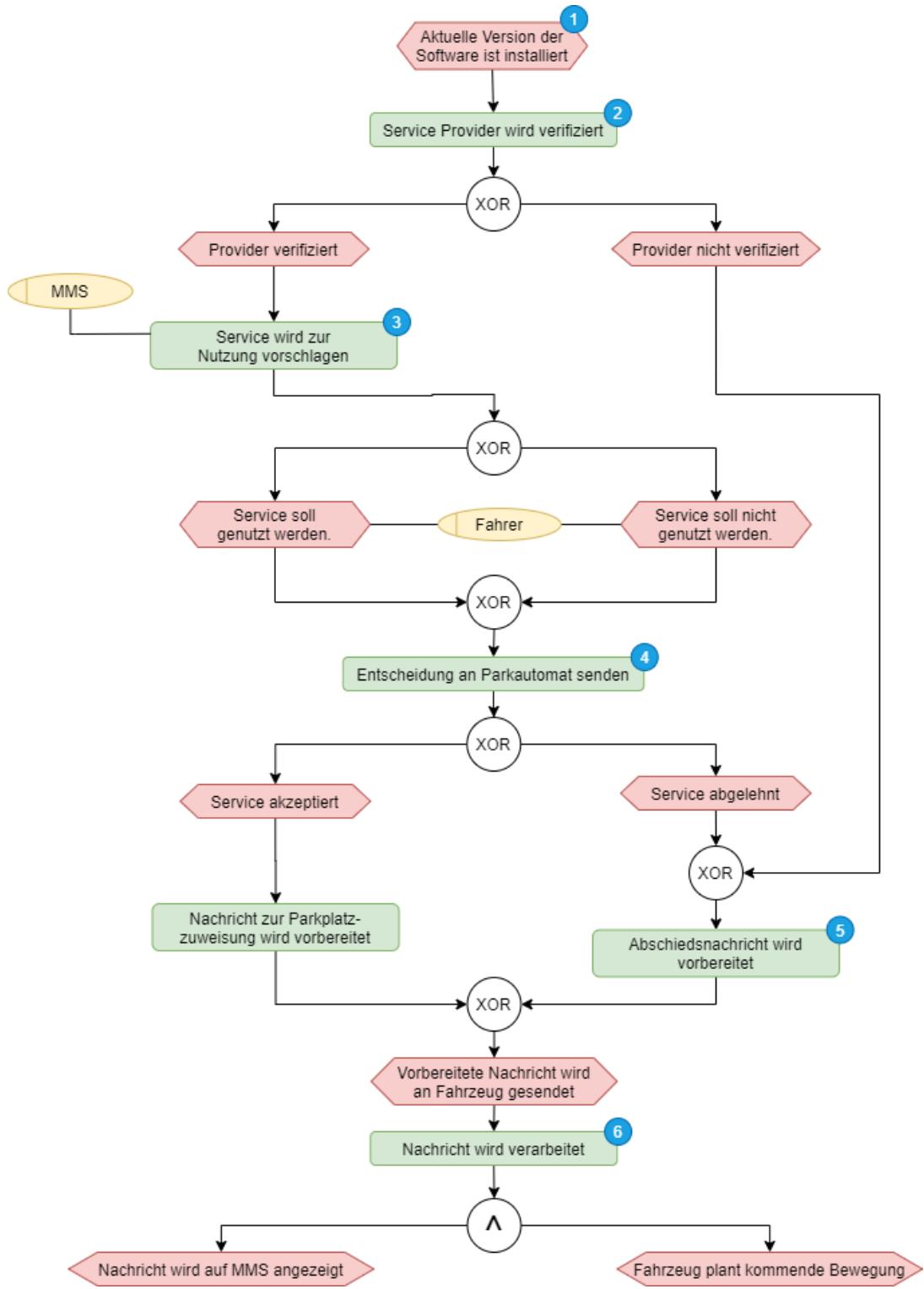


Abbildung 12: Nutzungsprozess von Software

#### **4.4 Ausblick**

Die in diesem Kapitel erstellten Konzepte wurden mit der Intention entwickelt, einzelne Bausteine der Wertschöpfungskette zu detaillieren und im Prototypen implementiert zu werden. So stellt die Identifikation von Berechtigungen für einzelner Systeme eines Fahrzeugs eine geeignete Grundlage für die Sicherheit intelligenter Fahrzeuge. Fahrzeughalter können durch diese und weitere Kennzahlen bessere Kaufentscheidungen treffen und ihr Fahrzeug so erweitern, dass es ihren Ansprüchen entsprechend aufgebaut ist. Der aufgeführte Suchalgorithmus hat gezeigt, wie die Aufgaben des Bausteins „*Automatische Erkennung von Softwarebedarf*“ durchgeführt werden können. Die Kommunikationsprotokolle veranschaulichen den Nachrichtenfluss zwischen den einzelnen Modulen des Prototypen *Server*, *Fahrzeug* und *Service Provider* und integrieren Aufgaben der Bausteine „*Automatische Erkennung von Softwarebedarf*“, „*Angebotsunterbreitung*“ und „*den gesicherten Download über das Internet*“. Das kommende Kapitel stellt den Prototypen vor, in welchem das zweite und dritte Kommunikationsprotokoll implementiert wurden.

## 5 Vorstellung des Prototypen

Der Prototyp demonstriert, wie eine Software entsprechend der in Kapitel 3.2 aufgeführten Bausteine einem Fahrzeughalter bereitgestellt und von diesem genutzt werden kann. Konkret wird die Bereitstellung einer Software dargestellt, mit welcher **ein Fahrzeughalter ein Parkticket kaufen und das Fahrzeug anschließend selbstständig zum Parkplatz fahren kann**. Zunächst wird in Kapitel 5.1 dargestellt, wie der Prototyp installiert werden kann. Im Anschluss daran wird in Kapitel 5.2 die Software Architektur des Prototypen vorgestellt. Abschließend werden die Funktionen des Prototypen vorgestellt und wie diese genutzt werden können.

### 5.1 Architektur des Prototypen

Abbildung 13 zeigt die Grundlegende Architektur des Prototypen. Der OEM-Server ist das Back-End des Shops und stellt die Single-Source-Of-Truth für Fahrzeuge dar. Der **Software Applikation Manager (SAM)**, die **Mensch Maschine Schnittstelle (MMS)** und die **Carla Simulation** bilden gemeinsam die Repräsentation des simulierten Fahrzeugs. Der Software Applikation Manager ist mit allen Modulen des Prototypen verbunden und stellt somit die zentrale Steuerungseinheit des Fahrzeugs dar. Über ihn erfolgt die Installation und die Nutzung von Softwares. Die Mensch Maschine Schnittstelle ist eine Android-Schnittstelle, auf welcher eine Version des Software Shops als App installiert ist. Über diese App können Nachrichten von dem Software Applikation Manager empfangen und verarbeitet werden und es können Nutzereingaben getätigter werden, durch welche der Ablauf der Simulation gesteuert werden kann. Alle Drei Module sind in Java implementiert worden. Die Python-basierte Carla Simulation ist die visuelle Repräsentation des Fahrzeugs und veranschaulicht getroffene Entscheidungen.

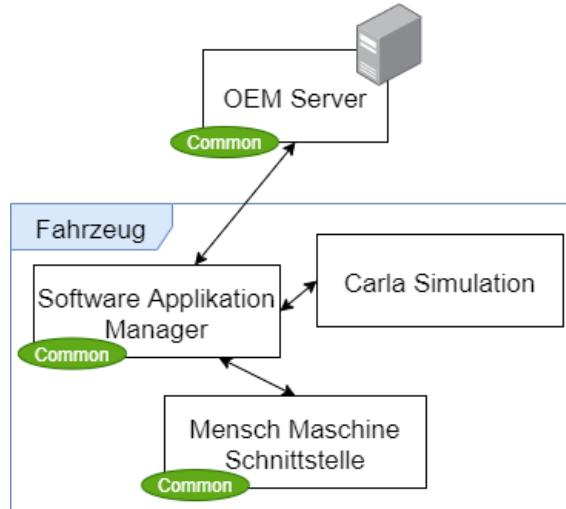


Abbildung 13: Grundlegende Architektur der Prototypen

Der SAM, der Server und die MMS sind alle Java-basiert und kommunizieren auf Basis der **Common-Library** miteinander, welche in Abbildung 14 gezeigt wird. Sie ist eine Sammlung an Java-Objekten, welche für die Anwendungsfall-bezogene Kommunikation notwendig sind und daher jedem Modul bekannt sein müssen. Sie enthalten Primitive Werte (*Integer, String, Long, etc.*) und leiten alle vom Serializable-Interface ab, damit sie über geschaffenen Netzwerkschnittstellen verschickt werden können.

Die Common-Objekte sind in zwei Gruppen unterteilt, die **Environment-Objekte** und **Messages**. Messages sind die Objekte, die im Prototypen zwischen den einzelnen Komponenten verschickt werden und so den Ablauf des Prototypen bestimmen. Sie beinhalten neben Primitiven Datentypen auch Environment Objekte und stellen die Basis der anwendungsfallbezogenen Kommunikation dar. Es wird zwischen SoftwareMessages und ServiceMessages unterschieden, wobei SoftwareMessages Nachrichten sind die zur Installation oder Nutzung einer Software benötigt werden und Service-Messages sind jene, die zur Kommunikation mit einem Service Provider notwendig sind.

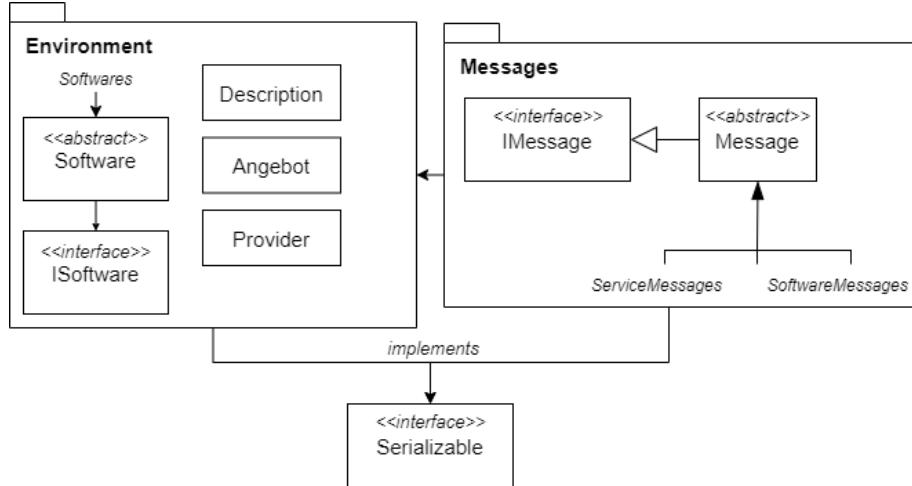


Abbildung 14: Aufbau der Common Library

Die in den Messages enthaltenen Environment-Objekte vereinfachen die Kommunikation der Module im Kontext des Anwendungsfalls. Die *Description* wird verwendet, wenn eine Software oder ein Service von einer Nachricht beschrieben werden soll und das *Provider*-Objekt ist dafür da, sowohl Software- als auch Service Provider einbeziehen zu können. Das *Angebot*-Objekt wird für die Bereitstellung von Auswahlmöglichkeiten beim Kauf einer Software wie zum Beispiel Kauf und Abo verwendet. Die abstrakte Klasse *Software* ist die Elternklasse für Softwares, welche dem Prototypen neue Anwendungsfälle hinzufügen sollen. Jede neu hinzugefügte Software überschreibt die von dieser Klasse vorgegebene Methode *handleMessage()*. Diese Methode wird im Prototypen aufgerufen, um Nachrichten von einer Services gemäß des Anwendungsfallkontexts zu verarbeiten.

### 5.1.1 Architektur des Software Applikation Managers (SAM)

Wie zuvor erwähnt, wird über den Software Applikation Manager das Geschehen des Prototypen gesteuert. Er ist, wie in Abbildung 15 zu sehen ist, in drei Module aufgeteilt: Die GUI ist die View-ebene des Prototypen, über welche das Geschehen der Simulation nachvollzogen und gesteuert werden kann. Die GUI wurde mit dem JavaFX-Framework erstellt und es wurde AfterburnerFX als weitere Unterstützung hinzugenommen. Durch AfterburnerFX wird das View-binding der einzelnen Elemente der GUI automatisiert und somit viel „Boilerplate Code“ vermieden.

Im *Car* befindet sich der MessageHandler, welcher die zentrale Steuereinheit des SAM ist. Er verbindet sich mit Hilfe der Klassen des *Networks* mit dem Server und öffnet die Ports, mit welchen sich die MMS und die Carla-Simulation verbinden können. Über diese Schnittstellen eingehende Nachrichten werden auf einen EventBus gepostet, der sich im MessageHandler befindet. Hierdurch kann die Simulation auf die Nutzereingaben aus GUI und MMS reagieren und die Nachrichten getrennt voneinander behandeln. Für die Installation und Nutzung von Softwares wird der SoftwareManager

genutzt, welcher alle installierten Softwares des Fahrzeugs verwalten und zum benötigten Zeitpunkt bereitstellen kann.

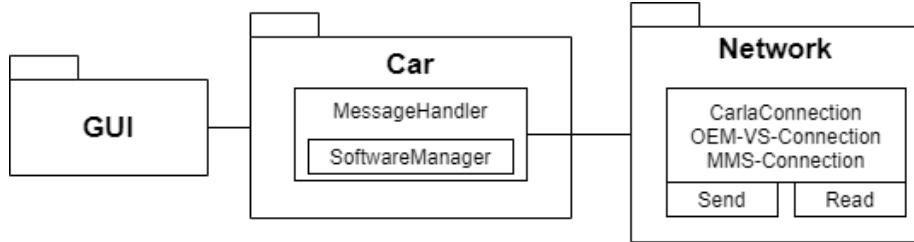


Abbildung 15: Architektur des Software Applikation Managers

### 5.1.2 Architektur des OEM-Servers (Server)

Die in Abbildung 16 dargestellte Architektur wurde nach der Vorlage des Uptane-Standards entworfen. Es wird eine Netzwerkverbindung mittels Netty<sup>20</sup> erstellt, mit welcher sich der SAM verbinden kann. Durch die Integration von Netty ist es möglich mehrere Software Applikation Manager mit dem Server zu verbinden, wodurch der Prototyp einfach skaliert werden kann. Über Netty eingehende Nachrichten werden auch hier wieder über einen EventBus an den Director weitergeleitet und von diesem verarbeitet. Der Director ist mit dem **Image Repository** verbunden, in welchem sich die Softwares befinden die im Shop verfügbar sind. Uptane gibt vor, zusätzlich eine **Inventory Database** zu führen in welcher die Manifeste aller im Shop registrierter Fahrzeuge in der Form gespeichert sind wie sie zuletzt von dem Director verifiziert wurden, [22, Vgl. Abschnitt 5.3.2.2] welcher daher auch hier integriert wurde. Wird eine Anfrage an den Server geschickt, vergleicht der Director das mitgeschickte Manifest mit dem in der Inventory Database gespeicherten Version. Werden Unterschiede oder sonstige Sicherheitslücken festgestellt, wird die Anfrage des Fahrzeugs für neue Software zurückgewiesen. [22, Vgl. Abschnitt 5.3.2.1]

Der Director kann eingehende Anfragen an den Software-User-Pattern-Recognizer (*SUPR*) weiterleiten, welcher in Kapitel zwei vorgestellt wurde. Der SUPR ist dafür zuständig, den Software-Bedarf von Fahrzeughaltern anhand ihrer Fahrdaten zu bestimmen. Er hat ebenfalls Zugriff auf das Image Repository und das *Scenario Repository* in welchem, wie in Kapitel zwei erwähnt, die zu einer Software hinzugefügten Szenarien gespeichert sind. Stellt der SUPR einen Softwarebedarf fest, wird dieser an den Director weitergegeben der infolgedessen die entsprechende Software dem Fahrzeughalter vorschlägt.

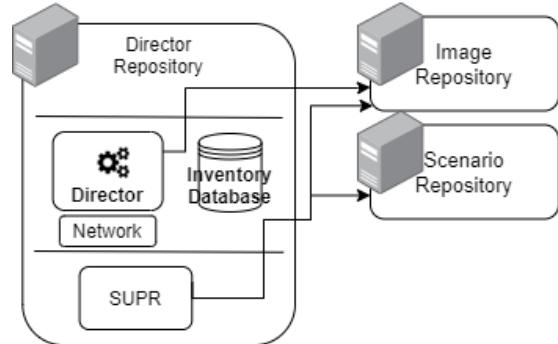


Abbildung 16: Architektur des Servers

<sup>20</sup><https://netty.io/>

## 5.2 Installation

Die Installation der einzelnen Module muss getrennt voneinander passieren. Für die vollständige Nutzung muss sichergestellt sein, dass Carla auf dem PC auf welchem der Prototyp installiert werden soll lauffähig ist. Schauen Sie hierzu in die Systemanforderungen von Carla.<sup>21</sup> Der SAM und der Server müssen auf dem selben PC installiert werden, während die anderen Module können auch auf weiteren Geräten genutzt werden können. Hierzu müssen die IP-Adressen und Ports der Module über deren Config-Dateien angepasst werden. Wie dies getan werden kann und welche weiteren Schritte für die Installation befolgt werden müssen, stellt die folgende Anleitung dar. Sie ist für Windowsgeräte ausgelegt und dauert ca. 30-40 Minuten.

### 5.2.1 Server und SAM Installation

#### 1. .zip-Ordner Herunterladen

Gehen Sie auf <https://github.com/hesty98/bachelorthesis>. Klicken Sie auf den "releases"-Tab, welcher unter der Beschreibung des Repository zu finden ist. Laden Sie die Datei "prototyp.zip" herunter und entpacken Sie den Ordner auf Ihrem Laptop. In diesem Ordner finden Sie die .jar-Dateien mit welchen der Server und der SAM gestartet werden können. Der Ordner "PythonScriptsThesis" wird später für die Installation von Carla benötigt.

#### 2. IP-Adresse aufschreiben

Um das MMS und auch Carla mit dem Software Applikation Manager zu verbinden, müssen Sie Ihre IP-Adresse kennen. Öffnen Sie hierzu Ihr Windows-Terminal und geben Sie `ipconfig` ein. In der Zeile „IPv4-Adresse“ steht Ihre die IP-Adresse. Diese müssen Sie sich merken.

### 5.2.2 MMS Installation

#### 1. Android Studio herunterladen

Sie haben zwei Optionen, wie die Mensch Maschine Schnittstelle genutzt werden kann. Entweder sie simulieren das Android Gerät auf Ihrem PC oder Sie installieren die App auf ein eigenes Tablett/Smartphone. Für beides wird eine aktuelle Version von Android Studio benötigt, die Sie hier (<https://developer.android.com/studio>) herunterladen können. Installieren Sie Android Studio und fahren Sie anschließend vor.

#### 2. Code importieren

Das Repository der Mensch Maschine Schnittstelle liegt auf GitHub. Über Android Studio können Sie das Projekt direkt importieren. Klicken Sie hierfür auf „File“-> „new“-> „Project from Version control“-> „Git“ und geben Sie folgende URL ein: <https://github.com/hesty98/HMI.git>. Alternativ kann das Projekt auch als .zip-Datei heruntergeladen werden. Der extrahierte Ordner kann in Android Studio als Projekt geöffnet werden („File“-> „open“-> „Ordner wählen“).

#### 3. MMS Konfigurieren

Öffnen Sie in Android Studio die Datei Connection.java und passen Sie den Host entsprechend der IP des *Software Applikation Managers* an. Die Datei befindet sich im 'network'-Ordner (`app -> java -> com.linushestermeyer.hmi -> network`).

---

<sup>21</sup>[https://carla.readthedocs.io/en/latest/start\\_quickstart](https://carla.readthedocs.io/en/latest/start_quickstart), Aufgerufen am 21. Juni 2020

## 4. MMS aufsetzen

Sie können die Mensch Maschine Schnittstelle nun entweder auf einem eigenen Gerät installieren oder das Gerät auf ihrem PC simulieren. Für beide Optionen müssen Sie zuerst eine Konfiguration für die App erstellen. Folgen Sie hierzu folgender Anleitung: <https://developer.android.com/studio/run/rundbgconfig>.

### 4.1 Eigenes Tablet/Smartphone

Zur Installation auf einem eigenen Gerät müssen Sie zunächst die *Entwickler-Optionen* für dieses Freischalten und *USB-Debugging* aktivieren. Folgen sie hierfür folgender Anleitung: <https://mobilsicher.de/ratgeber/usb-debugging-aktivieren>. Wurde dies getan, kann das Gerät an den PC angeschlossen werden. Über den grünen 'Play'-Button in der oberen Menuleiste kann die App auf Ihrem Gerät installiert werden.

### 4.2 Virtual Device

Wenn Sie Die App **nicht** auf einem eigenen Gerät installieren wollen, müssen sie mit Android Studio ein Simulierte Gerät erstellen. Dies funktioniert nur, wenn ihr PC eine **Intel-CPU** hat, da Android Studio den *HAXM-Service* von Intel zur Simulation nutzt. Ist dies der Fall, können Sie mit Hilfe folgender Anleitung ein Virtuelles Gerät erstelle: <https://developer.android.com/studio/run/managing-avds>. Erstellen Sie eine Instanz eines Tablets nicht die eines Smartphone, damit die Benutzeroberfläche der MMS gut dargestellt wird.

Haben Sie ein *VD* erstellt, können sie anschließend über den grünen 'Play'-Button in der oberen Menuleiste die App auf diesem Gerät installieren.

### 5.2.3 Carla Installation

#### 1. Python installieren und SYS\_PATH hinzufügen

Damit Carla auf dem PC laufen kann, muss Python in Version 3.7 auf dem PC installiert sein und dem SYSTEM\_PATH hinzugefügt werden. Hierzu kann folgende Anleitung genutzt werden, wobei die „3.4“ durch eine „3.7“ zu ersetzen ist: <https://geek-university.com/python/add-python-to-the-windows-path/>. Der SYSTEM\_PATH kann bei der Installation auch durch das ankreuzen einer Checkbox automatisch gesetzt werden.

#### 2. Carla Herunterladen

Carla kann als .zip-Ordner hier (<https://github.com/carla-simulator/carla/blob/master/Docs/download.md>) heruntergeladen werden - nutzen Sie die Version 0.9.7. Die Dateien müssen in einen eigenen Ordner auf dem PC extrahiert werden. Nun müssen die für die Simulation notwendigen Python-Packages *pip*, *pygame* und *numpy* installiert werden. Die pip-Installation kann hier nachgelesen werden: <https://pypi.org/project/pip/>. Für die Installation von pip sollte die Administrator-Konsole verwendet werden. Suchen Sie hierzu im Windows Suchbalken nach „cmd“ und führen Sie das Terminal mittels eines Rechtsklick „als Administrator aus“. Die Installation von pip muss im 'PythonAPI'-Ordner des Carla-Downloads erfolgen, daher müssen Sie in der Konsole in diesen Ordner wechseln (*cd ..Pfad/Von/Carla/.../PythonAPI*). Ist pip installiert, können mittels folgender Zeile die übrigen packages hinzugefügt werden:

```
py -3.7 -u pip install --user pygame numpy
```

Im Anschluss daran sollten Sie Carla ein mal testen. Starten Sie zunächst die Carla.exe und wechseln Sie im Terminal in den PythonAPI/examples Ordner (*cd PythonAPI/examples*). Führen Sie anschließend folgenden Befehl durch:

```
py -3.7 spawn_npc.py
```

Klappt dies nicht, haben Sie bei der Installation einen Fehler gemacht. Überprüfen Sie, ob die benötigten Python-Packages installiert sind und gehen Sie sicher, dass Python in der Version 3.7 dem SYSTEM\_PATH hinzugefügt wurde.

### 3. Skripte einfügen

In der extrahierten .zip-Datei des SAM und Servers befindet sich ein Ordner mit dem Namen *CarlaScriptThesis*. Kopieren Sie diesen Ordner in den *PythonAPI*-Ordner von Carla und Sie haben es geschafft!

### 4. Carla Konfigurieren

Abschließend sollten sie noch Konfigurationen an Carla vornehmen. Zunächst wird die Stadt geändert in welcher sich das Szenario abspielen wird. Öffnen Sie hierzu die „DefaultEngine.ini“-Datei aus dem Ordner „CarlaUE4/Config“, und ersetzen Sie *Town0X.Town0X* überall mit *Town02.Town02*. Löschen Sie die anderen Zeilen **nicht!**

```
EditorStartupMap=/Game/Carla/Maps/Town02.Town02  
GameDefaultMap=/Game/Carla/Maps/Town02.Town02  
ServerDefaultMap=/Game/Carla/Maps/Town02.Town02  
TransitionMap=/Game/Carla/Maps/Town02.Town02
```

Haben Sie die entsprechenden Werte eingetragen, speichern und schließen Sie die Datei. Anschließend wechseln sie in den Ordner *PythonAPI/CarlaScriptThesis* und editieren Sie die *simulation\_handler\_automatic.py*. Passen Sie die IP im Skript an die des **Software Applikation Managers** an.

## 5.3 Nutzung des Prototypen

Die Module des Prototypen müssen in bestimmter Reihenfolge gestartet werden. Öffnen Sie zunächst zwei Terminals im heruntergeladenem Ordner und eines *CarlaScriptThesis*-Ordner.

### 1. Server starten

```
java -jar Server.jar
```

### 2. Software Applikation Manager

```
java -jar SAM.jar
```

### 3. Carla

Starten Sie die CarlaUE4-Anwendung. Führen Sie anschließend im Terminal im Ordner „*PythonAPI/CarlaScriptThesis*“ das Skript *simulation\_handler\_automatic.py* aus.

```
py -3.7 simulation\_handler\_automatic.py
```

### 4. Mensch Maschine Schnittstelle

Starten Sie die App auf ihrem Gerät oder in dem Virtual Device. Gehen Sie sicher, dass die App zuvor geschlossen war.

Im folgenden haben Sie fünf Bildschirme, über die alle Module des Prototypen sichtbar sind. Die Mensch Maschine Schnittstelle wartet auf eingehende Nachrichten, ebenfalls wie Carla. Die GUI des Software Applikation Managers ist in drei Bereiche aufgeteilt. Der Log auf Linken Seite stellt alle Ereignisse aus Sicht des Fahrzeugs dar. Der Log auf der Rechten Seite der GUI stellt sämtliche Ereignisse dar, die von dem Parkautomaten wahrgenommen werden. In der unteren rechten Ecke sind diverse Knöpfe zu sehen mittels welcher der Prototypen gesteuert werden kann - Abbildung 17 zeigt einen Screenshot dieser. In der Carla Simulation können Sie die Kamera mit Hilfe der WASD-Tasten und der Maus bewegen. Wenn Sie sich geradeaus und dann links bewegen, kommen sie zu dem 'Parkplatz'.



Abbildung 17: Steuerung des Prototypen über die GUI

Über die oberen Knöpfe wird der Ablauf der Simulation gesteuert. Es ist immer nur möglich einen einzelnen Knopf zu bestätigen, die restlichen sind verblasst.

**Fahrzeug in Carla spawnen** Bei Bestätigung des Knopfes wird ein Fahrzeug in Carla gespawnt. Dies kann je nach Netzwerkgeschwindigkeit ein wenig dauern.

**Zum Parkplatz fahren** Bei Bestätigung des Knopfes fährt das Fahrzeug geradeaus in Richtung des Parkplatzes und hält vor diesem.

**In Registrierungszone fahren** Bei Bestätigung des Knopfes fährt das Fahrzeug in die *Registrierungszone* des Parkplatzes und hält in dieser an.

**Beim Fahrzeug registrieren** Bei Bestätigung des Knopfes schickt der Service Provider (*Parkautomat*) eine Nachricht an das Fahrzeug. Die Nachricht wird im Log dargestellt und über die Mensch-Maschine-Schnittstelle kann auf diese reagiert werden.

**Dem Fahrzeug Anweisungen geben** Bei Bestätigung des Knopfes wird dem Fahrzeug ein Anweisung gegeben, wohin es fahren soll. Entweder das Fahrzeug entfernt sich vom Parkplatz oder es parkt auf diesem.

**Neustart** Bei Bestätigung des Knopfes wird das Fahrzeug von der Karte entfernt und die Simulation kann erneut durchgeführt werden.

Die unteren Knöpfe bieten die Möglichkeit die Simulation anderweitig zu beeinflussen. Der in Rot

dargestellte Indikator „*Software nicht installiert*“ gibt an, ob die Parksoftware bereits installiert ist oder nicht.

**Fahrzeug gehackt** Bei Bestätigung des Knopfes wird das Manifest des Fahrzeugs geändert („*gehackt*“). Wenn die Software noch nicht installiert ist (sie Indikator), kann sie auch dann nicht installiert werden weil das der Server die Installationsanfrage zurückweist. Die erneute Bestätigung des Knopfes macht dies rückgängig.

**SW-Vorschläge** Bei Bestätigung des Knopfes wird verhindert, dass auf der MMS Softwares vorgeschlagen werden. Das erneute bestätigen macht dies rückgängig. Dies soll die Funktionalität eines *Fahrzeug-basierten SUPR* darstellen.

**Service Provider verifiziert** Die Bestätigung des Knopfes hat zur Folge, dass der Parkservice nicht mehr genutzt werden kann da der Service Provider nicht verifiziert ist. Das erneute bestätigen macht dies rückgängig.

Auf der Mensch-Maschine-Schnittstelle werden PopUps angezeigt, wenn eine Software installiert oder ein Service genutzt werden soll. In diesen Popups kann ein Angebot ausgewählt werden und der Kauf/die Nutzung von Services und Softwares akzeptiert oder abgelehnt werden. Es sind die Grundlagen implementiert, die es künftige erlauben den Shop auch als solchen zu nutzen - diese haben in der Arbeit jedoch keinen weiteren Einfluss mehr gefunden.

## 5.4 Analyse des Prototypen und Ausblick auf die Weiterentwicklung

Im Prototypen wurde die im Forschungsseminar erläuterte Uptane Architektur versucht grundlegend zu implementieren. Hierdurch konnte gezeigt werden, wie ein Fahrzeug in der Zukunft von externen Eingriffen geschützt und die Sicherheit so gewährleistet werden kann. Der Prototyp hat dargestellt, wie eine Software über eine kabellose Schnittstelle orts- und zeitunabhängig gekauft und installiert werden kann. Zusätzlich wurde die Interaktion mit Service Providern veranschaulicht. Über die GUIs kann nachvollzogen und gesteuert werden, was in der Simulation passieren sollen. Die visuelle Repräsentation des Fahrzeugs in Carla unterstützt die Wirkung des Prototypen enorm, da der Betrachter direkt die Auswirkungen seiner Entscheidungen sieht. Auch die Interaktion über die Android-basierte Mensch Maschine Schnittstelle hat sich als sinnvoll erwiesen, da hierdurch ein guter Bezug zur Realität und Android Automotive gezogen werden kann.

Die erstellte Software Architektur lässt eine einfache Erweiterung des Prototypen zu. Für einen neuen Anwendungsfall muss lediglich eine neue Software erstellt werden und das Carla Skript muss angepasst werden was in geringer Zeit möglich ist. Die Nachrichten (*Messages*) wurden generisch gehalten, damit sie künftig für neue Anwendungsfälle verwendet werden können.

Ein Bug, der sich nicht ohne weiteres beheben lies, ist die Dysfunktion des Carla-Skripts auf unterschiedlichen Computern. Der Bug ist, dass das Fahrzeug in Carla auf unterschiedlichen Computern unterschiedlich schnell fährt und lenkt, wodurch nicht immer der Parkplatz erreicht werden kann. Um den Prototypen dennoch komplett sehen zu können, wurde ein Video auf YouTube hochgeladen: <https://www.youtube.com/channel/UC02segbDQ5BnGwshmWXeRcA>

Der Prototyp bietet noch eine Vielzahl an Erweiterungsmöglichkeiten. Der **Server** könnte um weitere Uptane-Funktionen erweitert werden und durch die Erstellung eines Server-seitigen SUPR wird die Möglichkeit geschaffen Suchalgorithmen wie sie im Laufe der Arbeit vorgestellt wurden in den Prototypen zu integrieren. Auch der **Software Applikation Manager** kann durch diverse neue Funktionen Sinnvoll erweitert werden. So kann ein Fahrzeug-basierter SUPR, dargestellt in Kapitel 2.3.3, bestimmen wann ein PopUp auf der MMS erscheinen soll und so das wahllose erscheinen von Softwarevorschlägen verhindern, was aktuell über den **SW-Vorschläge**-Knopf getan wird. Die Steuerung über die GUI sollte den neuen Anwendungsfällen entsprechend angepasst werden, da nicht in jedem Anwendungsfall eine Kommunikation mit einem Service Provider stattfindet. Die erstellten Knöpfe könnten auch generiert werden, wenn dem Prototypen ein „Anwendungsfall“-Objekt hinzugefügt wird, anhand wessen sich die GUI anpassen wird. Entfernt man sämtliche Nutzereingaben des Prototypen und automatisiert so die gesamte Simulation, können die implementierten Anwendungsfälle beliebig hoch skaliert werden, wodurch die Zuverlässigkeit des Shops getestet werden könnte.

Im Laufe der Entwicklung wurde deutlich, dass Carla äußerst umfangreich ist. Würden die Funktionen von Carla ausgereizt werden, könnte eine Vielzahl neuer Anwendungsfälle zum Prototypen hinzugefügt und in Carla visuell dargestellt werden. Auch ein Wechsel zwischen autonomen Fahrfunktionen und eigenständiger Steuerung des Fahrzeugs kann in Carla erfolgen. Im aktuellen Prototypen ist Carla nur eine visuelle Repräsentation, wodurch eine zusätzliche Kommunikationsebene entsteht und die Laufzeit des Prototypen verschlechtert wird. Durch eine Implementierung des SAM in Python kann dies verbessert werden und die Architektur kann um ein Modul verkleinert werden, was auch die Installation erleichtern würde. Carla wäre dadurch sowohl die visuelle Repräsentation des Fahrzeugs als auch die "Business Logik".

## 6 Reflexion der Ergebnisse

Die zu Beginn der Bachelorarbeit aufgestellte Forschungsfrage stellte die Anforderung, wichtige Bausteine einer Wertschöpfungskette für die Erkennung und Bereitstellung neuer Fahrfunktionen zu identifizieren und darzustellen, wie diese von Automobilherstellern umgesetzt werden können. Das erstellte Business Model Canvas wurde mit der Intention erstellt, als Einführung in den Markt für Fahrfunktionssoftware zu dienen. Die hierdurch zusätzlich entstandenen Inhalte haben eine sehr gute Grundlage für einen "Blick über den Tellerrand" geboten, welcher eine mögliche Zukunft von Fahrzeugen veranschaulicht.

Die in der Wertschöpfungskette identifizierten Bausteine haben viele Aufgaben eines Software Shops aufgedeckt. Einige der Bausteine wurden in dieser Arbeit aus Platzgründen nicht im Detail erläutert, damit die im Fokus stehenden Bausteine der Bedarfserkennung und Bereitstellung ausreichend detailliert werden konnten. In den Kapiteln 2 und 4 wurden technischen Konzepte erstellt um diese Bausteine tiefer gehend zu betrachten. Die Ergebnisse dessen sind zufriedenstellend, da sie eine guten Überblick über die verschiedenen Aufgabenfelder der Erkennung und Bereitstellung von Software bieten. Dennoch hätten einige dieser Bausteine qualitativ besser erörtert werden können, zum Beispiel durch die Integration von mehr Bausteinen in den Prototypen. Dieser hat letzten Endes die vier folgenden Bausteine der Wertschöpfungskette visualisiert:

- Betreiben des Shops
- Automatische Erkennung von Softwarebedarf
- Angebotsunterbreitung
- (Sicherer) Download über das Internet

Durch die entwickelte Architektur können diesem jedoch weitere Bausteine und neue Anwendungsfälle mit nur geringem Zeitaufwand hinzugefügt werden. Die GUI des Fahrzeugs stellt die dort geschehenden Ereignisse gut dar und die Steuerung über Knöpfe ist zielführend, jedoch nur für den Anwendungsfall der Bachelorarbeit ausgelegt. Änderungen in der Architektur wie das Auslagern des Anwendungsfalls hätten die Erweiterung zusätzlich vereinfacht. Die Integration des Uptane-Standards in den Prototypen und die Einbindung von OpenScenario-Dateien in technische Konzepte hat die Wichtigkeit von OpenSource-Lösungen verdeutlicht für Fahrzeuge aufgezeigt.

Einige Ergebnisse wie zum Beispiel die Bausteine des Supply Chain Managements oder des generellen Betriebs haben keinen direkten Bezug auf die Forschungsfrage der Bachelorarbeit. Diese Problematik entstand durch eine Unterschätzung des Themas zu Beginn. Es wurde erst im Laufe der Erarbeitung deutlich, dass zum Verständnis einiger Bausteine der Bedarfserkennung und Bereitstellung weiteres Grundwissen notwendig ist, welches durch das Business Model Canvas vermittelt wurde. Die Integration dieser zusätzlichen Informationen hat letzten Endes zu einem besseren Gesamtergebnis beigetragen und viele Blickwinkel eröffnet wodurch mehrere Themen für Abschluss- und Forschungsarbeiten durch diese Arbeit vorgeschlagen werden können.

## 7 Ausblick

In direkter Anknüpfung an die Arbeit sollten die identifizierten Bausteine weiter erforscht und im Prototypen veranschaulicht werden. Es sollte eine GUI für den Server erstellt werden, damit auch

die dortigen Entscheidungen nachvollzogen werden können. Außerdem kann die Simulation in Carla durch weitere Verkehrsteilnehmer realistischer gestaltet werden. Durch die Implementierung des "SSoftware User Pattern Recognizer"(SUPR, Kapitel 2) des Fahrzeugs und des Servers würde den Prototypen vervollständigen. Ein Prototyp, der diese Anforderungen erfüllt und zudem mehrere Anwendungsfälle umfasst, kann sehr gut für die Vorstellung auf Messen oder anderweitigen Präsentationen genutzt werden, um "die Zukunft des Autofahrens" zu zeigen.

In der Arbeit wurde davon ausgegangen, dass Software auf einem einzelnen Fahrzeug installiert wird. Es ist vorstellbar, dass gekaufte Softwares nicht nur auf einem einzigen Fahrzeug genutzt werden sollen wozu vor allem der wachsende Car-Sharing Markt beiträgt. Damit Fahrer sich einfach auf einem einloggen können und im folgenden alle jemals von ihm gekauften Softwares verfügbar sind ist ein technisches Konzept zu erstellen. Dieses soll darstellen, wie Software nicht Fahrzeuggebunden sondern Fahrzeughalter-gebunden gekauft und genutzt werden kann.

Damit Softwares auf Fahrzeugen verschiedener, möglicherweise konkurrierender Automarken installiert und genutzt werden können muss eine Software-API konzipiert werden. Hierbei besteht die Schwierigkeit, dass selbst Fahrzeuge der selben Automarke unterschiedliche Software- und Hardware-Architekturen aufweisen und eine generalisierte Ansteuerung dieser Komponenten dementsprechend umfangreich gestalten kann.

## Literatur

- [1] (Aufgerufen am 10. Januar 2020) Andrea Rosenbusch. User-centered design in sieben punkten kurz erklärt. [https://zeix.com/wp-content/uploads/2011/04/ict-jb2011\\_artikel\\_zeix\\_rosenbusch\\_felix\\_final.pdf](https://zeix.com/wp-content/uploads/2011/04/ict-jb2011_artikel_zeix_rosenbusch_felix_final.pdf), 2011.
- [2] (Aufgerufen am 10. Januar 2020) Divya Krishnan. Ui/ux for autonomous vehicle interface to build trust. <https://medium.com/divya-krishnan-design/ui-ux-for-autonomous-vehicle-interface-to-build-trust-de7f4c545c3b>, 2019.
- [3] (Aufgerufen am 10. Januar 2020) GrummelJS. Datei:walter e. boomer.jpg. [https://de.wikipedia.org/wiki/Datei:Walter\\_E.\\_Boomer.jpg](https://de.wikipedia.org/wiki/Datei:Walter_E._Boomer.jpg), 2008.
- [4] (Aufgerufen am 10. Januar 2020) Hans-Christian Reuss. Intelligente fahrzeuge. [https://www.uni-stuttgart.de/presse/archiv/themenheft/07/intelligente\\_fahrzeuge.pdf](https://www.uni-stuttgart.de/presse/archiv/themenheft/07/intelligente_fahrzeuge.pdf), 2010.
- [5] (Aufgerufen am 10. Januar 2020) Jeff Schneider. Ri seminar: Jeff schneider : Self driving cars and ai. [https://www.youtube.com/watch?v=jTio\\_MPQRYc&t=1497s](https://www.youtube.com/watch?v=jTio_MPQRYc&t=1497s), 2019.
- [6] (Aufgerufen am 10. Januar 2020) Juergen Daunis. The automotive industry in transformation – business model disruption. <https://www.ericsson.com/en/blog/2017/11/the-automotive-industry-in-transformation--business-model-disruption>.
- [7] (Aufgerufen am 10. Januar 2020) Linux Foundation. Uptane design. <https://uptane.github.io/design.html>, 2020.
- [8] (Aufgerufen am 10. Januar 2020) Marius Dupius. Open scenario. <https://www.asam.net/standards/detail/openscenario/>, 2015.
- [9] (Aufgerufen am 10. Januar 2020) Martin Seibert. Personas geben zielgruppen gesichter. <https://blog.seibert-media.net/blog/2008/09/12/personas-geben-zielgruppen-gesichter/>, 2008.
- [10] (Aufgerufen am 10. Januar 2020) Michael Green. Angry granpa. [https://commons.wikimedia.org/wiki/File:Angry\\_Grandpa\\_-\\_2015\\_\(cropped\).jpg](https://commons.wikimedia.org/wiki/File:Angry_Grandpa_-_2015_(cropped).jpg), 2015.
- [11] (Aufgerufen am 10. Januar 2020) Seobility. User centered design.
- [12] (Aufgerufen am 10. Januar 2020) Thomas Weißgerber. Konzepte und methoden des user centered design. [http://www.inf.fu-berlin.de/inst/ag-se/teaching/S-BSE/326\\_weissgerber\\_UCD.pdf](http://www.inf.fu-berlin.de/inst/ag-se/teaching/S-BSE/326_weissgerber_UCD.pdf), 2015.
- [13] (Aufgerufen am 10. Januar 2020) Tibchris. File:a beautiful girl in her black dress.jpg. [https://commons.wikimedia.org/wiki/File:A\\_beautiful\\_girl\\_in\\_her\\_black\\_dress.jpg](https://commons.wikimedia.org/wiki/File:A_beautiful_girl_in_her_black_dress.jpg), 2010.
- [14] (Aufgerufen am 10. Januar 2020) usability toolkit.de. usability-toolkit.de: Usability für webprojekte. <http://usability-toolkit.de/usability/usability-in-web-projekten/>, 2020.
- [15] (Aufgerufen am 19. Juni 2020) Andreas Ahlswede. Anzahl registrierter kraftfahrzeuge weltweit in den jahren 2005 bis 2015. <https://de.statista.com/statistik/daten/studie/244999/umfrage/weltweiter-pkw-und-nutzfahrzeugbestand/>, 2017.

- [16] (Aufgerufen am 19. Juni 2020) Werner Sammer. Der business model canvas: Dein geschäftsmodell kompakt. <https://ut11.net/de/blog/dein-geschäftsmodell-kompakt-der-business-model-canvas/>, 2019.
- [17] (Aufgerufen am 21. Juni 2020) L. Rabe. Durchschnittliche tägliche nutzungsdauer von sozialen medien weltweit in den jahren 2012 bis 2018. <https://de.statista.com/statistik/daten/studie/475072/umfrage/taegliche-nutzungsdauer-von-sozialen-medien/>.
- [18] (Aufgerufen am 21. Juni 2020) Manager-Wiki. Wertkette (nach porter). <http://www.manager-wiki.com/unternehmensanalyse/12-wertkette>.
- [19] (Aufgerufen am 21. Juni 2020) Rilind Elezaj. Autonomous cars: Safety opportunity or cybersecurity threat? <https://www.machinedesign.com/mechanical-motion-systems/article/21837958/autonomous-cars-safety-opportunity-or-cybersecurity-threat>.
- [20] (Aufgerufen am 21. Juni 2020) Startplatz. Business model canvas. <https://www.startplatz.de/startup-wiki/business-model-canvas/>.
- [21] (Aufgerufen am 21. Juni 2020) Torsten Klanitz. Wertschöpfungskette. <https://refa.de/service/refa-lexikon/wertschoepfungskette>.
- [22] (Aufgerufen am 21. Juni 2020) Uptane Alliance. Uptane standard for design and implementation. <https://uptane.github.io/papers/uptane-standard.1.0.1.html>.
- [23] Frauen und Jugend (Aufgerufen am 21. Juni 2020) Bundesministerium für Familie, Senioren. Work life balance. <https://www.bmfsfj.de/blob/95550/eb8fab22f858838abd0b8dad47cbe95d/work-life-balance-data.pdf>, August 2005.
- [24] William Sims Bainbridge. *Berkshire encyclopedia of human-computer interaction*, volume 1. Berkshire Publishing Group LLC, 2004.
- [25] Verband der Automobilindustrie. Automatisierung - von fahrassistentensystemen zum automatisierten fahren. *VDA Magazin - Automatisierung*, 2015.
- [26] Lex Fridman. 'human-centered autonomous vehicle systems: Principles of effective shared autonomy'. Technical report, Massachusetts Institute of Technology (MIT), 2018.
- [27] Thomas Künneth. *Android 8 - Das Praxisbuch für Java-Entwickler*.
- [28] Markus Reinke. *30 Minuten Verkaufspräzisierung*. Gabel Verlag, 2013.
- [29] R. Pokam S. Langlois S. Debernard, C. Chauvin. Designing human-machine interface for autonomous vehicles. Technical report, International Federation of Automatic Control, 2016.
- [30] Gerd-Inno Spindler. *Kaufverhalten und Kaufentscheidung*, pages 39–48. Springer Fachmedien Wiesbaden, Wiesbaden, 2016.
- [31] Niklas Stelter. Entwurf und simulation von monitoren für die evaluierung von updates intelligenter fahrzeuge. Master's thesis, CvO Universität Oldenburg, 2019.

## 8 Codeentnahmen

*Teile des Codes des Prototypen wurden folgenden Internetseiten, die im Code an entsprechenden Stellen angegeben sind, entnommen:*

<https://www.programcreek.com/java-api-examples/?class=javafx.application.Platform&method=runLater>

<https://netty.io/wiki/user-guide-for-5.x.html>

<https://stackoverflow.com/questions/30186343/passing-java-object-to-python>

[https://de.wikibooks.org/wiki/Java\\_Standard:\\_Socket\\_und\\_ServerSocket\\_\(java.net\)](https://de.wikibooks.org/wiki/Java_Standard:_Socket_und_ServerSocket_(java.net))

<http://afterburner.adam-bien.com/>

[https://docs.oracle.com/javafx/2/get\\_started/hello\\_world.htm](https://docs.oracle.com/javafx/2/get_started/hello_world.htm)